**Федеральное агентство связи**

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №7: Веб-Сканер

по дисциплине «Кроссплатформенные технологии программирования»

В этой лабораторной работе вам необходимо будет реализовать элементарный веб-сканер.

Выполнил: студент

группы БСТ1803

Корнеев Александр

Вариант 12

Москва 2020

**Содержание**

[1 Задание на разработку программы 4](#_Toc42187750)

[2 Разработанные функции и классы 6](#_Toc42187751)

[2.1 Класс Crawler 6](#_Toc42187752)

[2.1.1 Внутренний класс MyRunnable подкласс Runnable 6](#_Toc42187753)

[2.1.2 Метод void startMultiThreadCrawl 6](#_Toc42187754)

[2.1.3 Метод private void startCrawl 6](#_Toc42187755)

[2.1.4 Метод private boolean readURL 6](#_Toc42187756)

[2.1.5 Метод private boolean ParseNewURL 6](#_Toc42187757)

[2.1.6 Метод private boolean ParseNewURL 7](#_Toc42187758)

[2.2 Класс InfoURL 7](#_Toc42187759)

[2.2.1 Метод public void addQuoute 7](#_Toc42187760)

[2.2.2 Метод public void addHref 7](#_Toc42187761)

[2.2.3 Метод private void readDescription 7](#_Toc42187762)

[2.2.4 Метод private void readTitle 7](#_Toc42187763)

[2.2.5 Метод private void parseH 7](#_Toc42187764)

[2.2.6 Метод private void parseWord 8](#_Toc42187765)

[2.2.7 Метод private String readStyle 8](#_Toc42187766)

[2.2.8 Метод private void calculateIndex 8](#_Toc42187767)

[2.2.9 Метод public void readLine 8](#_Toc42187768)

[2.2.10 Метод public double getRepeatIndex 8](#_Toc42187769)

[2.2.11 Метод public double getHrefIndex 8](#_Toc42187770)

[2.2.12 Метод public double getHIndex 8](#_Toc42187771)

[2.2.13 Метод public double getContentIndex 9](#_Toc42187772)

[2.2.14 Метод public int getQuoteCount 9](#_Toc42187773)

[2.2.15 Метод public String getTitle 9](#_Toc42187774)

[2.2.16 Метод public String getDescription 9](#_Toc42187775)

[2.2.17 Метод public String getURL 9](#_Toc42187776)

[2.3 Класс DBWriteURL 9](#_Toc42187777)

[2.3.1 Метод public void conn 9](#_Toc42187778)

[2.3.2 Метод public void created 10](#_Toc42187779)

[2.3.3 Метод public void writeDB 10](#_Toc42187780)

[2.3.4 Метод public void updateQuoteDB 10](#_Toc42187781)

[2.3.5 Метод public void closeDB 10](#_Toc42187782)

[2.4 Класс SocketConnection 10](#_Toc42187783)

[2.4.1 Метод public boolean connect 10](#_Toc42187784)

[2.4.2 Метод void sendGET 10](#_Toc42187785)

[2.4.3 Метод private void getCode 11](#_Toc42187786)

[2.4.4 Метод public void close 11](#_Toc42187787)

[2.5 Класс URLDepthPair 11](#_Toc42187788)

[2.5.1 Метод String searchPattern 11](#_Toc42187789)

[2.5.2 Метод private void checkExtension 11](#_Toc42187790)

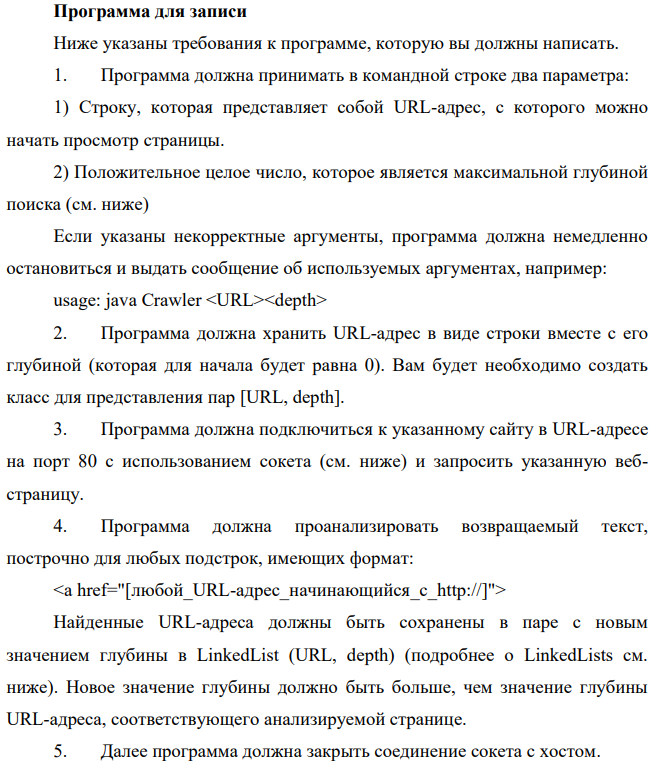
[2.6 Класс Web\_Scan 11](#_Toc42187791)

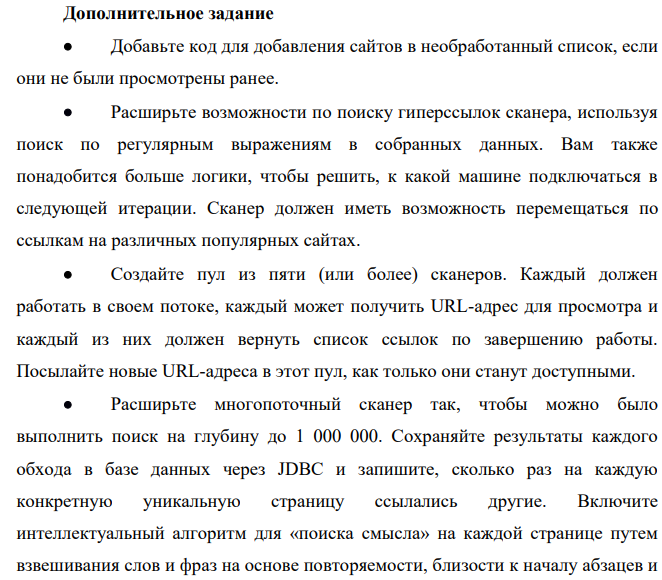
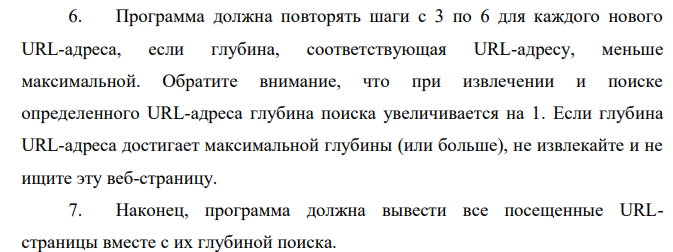
[2.6.1 Метод public static void main(String[] args) 12](#_Toc42187792)

[3 Код программы 12](#_Toc42187793)

[4 Результат работы программы 34](#_Toc42187794)

# Задание на разработку программы





# Разработанные функции и классы

## Класс Crawler

Класс реализующий сканер веб-страниц.

* + 1. Внутренний класс MyRunnable подкласс Runnable

Класс потока, содержит метод run, который начинается сканирование в отдельном потоке.

* + 1. Метод void startMultiThreadCrawl

Метод запускающий сканирование в несколько потоков.

* + 1. Метод private void startCrawl

Метод запускающий сканирование

* + 1. Метод private boolean readURL

Метод чтения url страницы

* + 1. Метод private boolean ParseNewURL

Метод парсинга строки на наличие уникальной url ссылки

* + 1. Метод private boolean ParseNewURL
  1. Класс InfoURL

Класс собирающий информацию о URL странице

* + 1. Метод public void addQuoute

Метод добавляющий количество "цитирования" данной страницы другими

* + 1. Метод public void addHref

Метод добавляющий количество ссылок на url на данной странице

* + 1. Метод private void readDescription

Метод занимающийся поиском описания страницы

* + 1. Метод private void readTitle

Метод занимающийся поиском заголовка title страницы

* + 1. Метод private void parseH

Метод занимающийся поиском заголовков на странице

* + 1. Метод private void parseWord

Метод занимающийся поиском повторяемости слов

* + 1. Метод private String readStyle

Метод для удаления тега style из строки или игнорирования данной строки если тег на ней не заканчивается

* + 1. Метод private void calculateIndex

Метод рассчитывающий индексы содержания страницы

* + 1. Метод public void readLine

Метод читающий передаваемую строку

* + 1. Метод public double getRepeatIndex

Метод возвращает “индекс повторяемости”

* + 1. Метод public double getHrefIndex

Метод возвращает “индекс ссылок”

* + 1. Метод public double getHIndex

Метод возвращает «индекс заголовков»

* + 1. Метод public double getContentIndex

Метод возвращает «индекс содержимого страницы»

* + 1. Метод public int getQuoteCount

Метод возвращает количество «цитирования» страницы

* + 1. Метод public String getTitle

Метод возвращает заголовок страницы

* + 1. Метод public String getDescription

Метод возвращает описание страницы

* + 1. Метод public String getURL

Метод возвращает url страницы

## Класс DBWriteURL

Класс для записи полученных URL и информации о них в базу данных SQLite3

* + 1. Метод public void conn

Метод осуществляющий подключение к БД

* + 1. Метод public void created

Метод осуществляющий создание таблицы

* + 1. Метод public void writeDB

Создание в базе данных записи о проверенной странице

* + 1. Метод public void updateQuoteDB

Обновление количества "Цитирования" данной страницы другими

* + 1. Метод public void closeDB

Метод закрывающий подключение к БД

## Класс SocketConnection

Класс осуществляющий работу с http запросами с помощью сокета

* + 1. Метод public boolean connect

Метод подключения к серверу

* + 1. Метод void sendGET

Метод отправляющий GET запрос на сервер

* + 1. Метод private void getCode

Метод, получающий код ответа

* + 1. Метод public void close

Метод закрывающий соединение с сокетом

## Класс URLDepthPair

Класс реализующий пару URL Depth

* + 1. Метод String searchPattern

Метод поиска подстроки соответствующей шаблону

* + 1. Метод private void checkExtension

Метод проверки на то, что расширение файла на конце найденного url или отсутствует или является допустимым, в противном случае выдает ошибку

## Класс Web\_Scan

Класс запускающий сканер Веб-Страниц

* + 1. Метод public static void main(String[] args)

Запускает сканирование по введенному адресу до введенной глубины поиска

# Код программы

/\*\*

\* Класс реализующий сканер веб-страниц

\*

\*/

**public** **class** Crawler

{

/\*\*

\* База данных, к которой ведется подключение для записи

\*/

**private** DBWriteURL db;

/\*\*

\* Список потоков

\*/

**private** LinkedList<MyRunnable> threadArray;

/\*\*

\* Шаблон регулярных выражений для поиска ссылок href

\*/

**static** **final** String ***HREF\_PATTERN*** = "(href=\").\*?\"";

/\*\*

\* Шаблон регулярных выражений для поиска тега a

\*/

**static** **final** String ***TAG\_A\_PATTERN*** = "(<a).\*?>";

/\*\*

\* Скомплированный шаблон поиска тега а

\*/

**private** **static** **final** Pattern ***TAG\_A\_PATTEN\_COMPILE*** = Pattern.*compile*(***TAG\_A\_PATTERN***);

/\*\*

\* Скомплированный шаблон поиска href

\*/

**private** **static** **final** Pattern ***HREF\_PATTERN\_COMPILE*** = Pattern.*compile*(***HREF\_PATTERN***);

/\*\*

\* Множество всех найденных адресов, для проверки уникальности нового домена

\*/

**private** HashMap<String,InfoURL> allURL = **new** HashMap<String,InfoURL>();

/\*\*

\* Множество всех найденных доменов, необходимый для "Умного" выбора новой страницы

\* для парсинка

\*/

**private** HashSet<String> allDomain = **new** HashSet<String>();

/\*\*

\* Список не проверенных, но найденны url адресов

\*/

**private** LinkedBlockingDeque <URLDepthPair> unCheckedList = **new** LinkedBlockingDeque <URLDepthPair>();

/\*\*

\* Список проверенных url адресов

\*/

**private** LinkedList <URLDepthPair> checkedList = **new** LinkedList <URLDepthPair>();

/\*\*

\* Список непроверяемых узлов

\*/

**private** LinkedList <URLDepthPair> noneCheckedList = **new** LinkedList <URLDepthPair>();

/\*\*

\* Максимальная глубина поиска

\*/

**private** **int** maxDepth;

/\*\*

\* Конструктор

\* **@param** url - Входящая url строка

\* **@param** maxDepth - Максимальная глубина поиска

\* **@throws** IOException

\* **@throws** UnknownHostException

\* **@throws** ClassNotFoundException

\*/

**public** Crawler(String url, **int** maxDepth) **throws** UnknownHostException, IOException, ClassNotFoundException, SQLException

{

URLDepthPair urlpair = **new** URLDepthPair(url,0);

unCheckedList.add(urlpair);

allURL.put(urlpair.url, **new** InfoURL(urlpair.url));

allDomain.add(urlpair.domain);

**this**.maxDepth = maxDepth;

db = **new** DBWriteURL();

}

/\*\*

\* Класс потока, при его инициализации начинается сканирование в отдельном потоке

\*/

**public** **class** MyRunnable **implements** Runnable {

/\*\*

\* Флаг, указывающий работает ли в данный момент поток

\*/

**public** **boolean** isRun = **false**;

**public** **void** run()

{

**while**(**true**)

{

**this**.isRun = **true**;

**try** {

startCrawl();

} **catch** (IOException | InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**this**.isRun = **false**;

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**boolean** flag = **true**;

**for**(MyRunnable var : threadArray)

{

**if**(var.isRun)

{

flag = **false**;

**break**;

}

}

**if**(flag) **break**;

}

}

}

/\*\*

\* Метод запускающий сканирование в несколько потоков

\* **@param** n - количество потоков

\* **@throws** SQLException

\* **@throws** ClassNotFoundException

\*/

**public** **void** startMultiThreadCrawl(**int** n) **throws** ClassNotFoundException

{

db.conn();

db.createDB();

**this**.threadArray = **new** LinkedList<MyRunnable>();

**for**(**int** i = 0;i<n;i++)

{

MyRunnable run = **new** MyRunnable();

**this**.threadArray.add(run);

Thread myThread = **new** Thread(run,"Thread" + i);

myThread.start();

}

**while**(**true**)

{

**try** {

Thread.*sleep*(5000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**boolean** flag = **true**;

**for**(MyRunnable var : threadArray)

{

**if**(var.isRun) flag = **false**;

}

**if**(flag) **break**;

}

db.closeDB();

}

/\*\*

\* Метод запускающий сканирование

\* **@throws** IOException

\* **@throws** SQLException

\* **@throws** InterruptedException

\*/

**private** **void** startCrawl() **throws** IOException, InterruptedException

{

**while**(!unCheckedList.isEmpty())

{

URLDepthPair urlpair = unCheckedList.pollFirst();

**if** (urlpair.depth<=maxDepth)

{

**if**(readURL(urlpair)) checkedList.addLast(urlpair);

System.***out***.println(urlpair.url + " " + urlpair.depth);

}

**else** noneCheckedList.add(urlpair);

}

}

/\*\*

\* Метод чтения url страницы

\* **@param** urlpair - содержит адрес url страницы

\* **@return** - true, если подключение удачно, иначе false

\* **@throws** IOException

\* **@throws** SQLException

\*/

@SuppressWarnings("finally")

**private** **boolean** readURL(URLDepthPair urlpair) **throws** IOException

{

InfoURL info = allURL.get(urlpair.url);

SocketConnection conn = **new** SocketConnection(urlpair);

// вернуть false, если соединение неудачно

**if** (!conn.connect()) **return** **false**;

conn.sendGET();

**if** (conn.code.equals("200"))

{

**try**

{

String line;

BufferedReader br = conn.br;

**while** (!(line = br.readLine()).equals("0"))

{

// Если строка содержит url, то увеличить количество ссылок

**if** (ParseNewURL(line,urlpair)) info.addHref();

info.readLine(line);

}

db.writeDB(info);

conn.close();

**return** **true**;

}

**finally**

{

conn.close();

**return** **false**;

}

}

// Если код ответа, не равен 200(OK)

**else** **return** **false**;

}

/\*\*

\* Метод парсинга строки на наличие уникальной url ссылки,

\* если таковая существует на заданной строке, то добавляет

\* новый url в необходимые элементы

\* **@param** line - Входящая строка

\* **@param** url - URLDepthPair по адресу которого была получена входящая строка

\* **@return** true, если строка содержит url, иначе false

\* **@throws** SQLException

\*/

**private** **boolean** ParseNewURL(String line, URLDepthPair url)

{

**try**

{

// Поиск тега а

Matcher m = ***TAG\_A\_PATTEN\_COMPILE***.matcher(line);

m.find();

String tagA = line.substring(m.start(),m.end());

// Поиск ссылки href в теге а

Matcher m2 = ***HREF\_PATTERN\_COMPILE***.matcher(tagA);

m2.find();

String href = tagA.substring(m2.start(),m2.end());

URLDepthPair newURL = **new** URLDepthPair(href,url.depth+1,url.domain);

// Увеличение количества "Цитирования страницы", если она уже есть в allURL

// Иначе добавление в allURL

**if** (allURL.containsKey(newURL.url))

{

InfoURL info = allURL.get(newURL.url);

info.addQuoute();

db.updateQuoteDB(info);

**return** **true**;

}

**else**

{

allURL.put(newURL.url, **new** InfoURL(newURL.url));

// Добавление в конец списка, проверяемых страниц, если это страница

// с не уникальным доменом, иначе в начало списка

**if** (allDomain.contains(newURL.domain)) unCheckedList.addLast(newURL);

**else**

{

unCheckedList.addFirst(newURL);

allDomain.add(newURL.domain);

}

**return** **true**;

}

}

**catch** (IllegalStateException e)

{

**return** **false**;

}

**catch** (MalformedURLException e)

{

**return** **false**;

}

}

/\*\*

\* **@return** checkedList - Список проверенных url

\*/

**public** LinkedList <URLDepthPair> getСheckedSites()

{

**return** checkedList;

}

/\*\*

\* **@return** noneCheckedList - Cписок непроверяемых url

\*/

**public** LinkedList <URLDepthPair> getUnСheckedSites()

{

**return** noneCheckedList;

}

/\*\*

\* Метод возвращающий информацию о сайте

\* **@param** udp - URLDepthPair содержащий url запрашиваемого сайта

\* **@return** - InfoURL данного сайта

\*/

**public** InfoURL getInfo(URLDepthPair udp)

{

**return** **this**.allURL.get(udp.url);

}

/\*\*

\* Метод возвращающий информацию о всех сайтах

\* **@return** - HashMap<String,InfoURL> содержащий информацию о всех найденных сайтах

\*/

**public** HashMap<String,InfoURL> getInfos()

{

**return** **this**.allURL;

}

}

/\*\*

\* Класс реализующий пару URL:Depth

\* URL - Адрес страницы url

\* Depth - Глубина поиска, на которой найдена данная страница

\*/

**public** **class** URLDepthPair

{

/\*\*

\* Индекс начала ссылки в в строке вида 'href="URL" ' (С любым символом на конце после кавычек)

\*/

**public** **static** **final** **int** ***START\_HREF*** = 6;

/\*\*

\* Количество символов до конца строки от конца ссылки в строке вида 'href="URL" ' (С любым символом на конце после кавычек)

\*/

**public** **static** **final** **int** ***HREF\_TOEND*** = 1;

/\*\*

\* Расширение html файла

\*/

**public** **static** **final** String ***HTML\_EXTENSION*** = ".html";

/\*\*

\* Допустимые расширения

\*/

**public** **static** **final** String ***ACCEPTABLE\_EXTENSION\_PATTERN*** = "(.html?)|(.php)";

/\*\*

\* Префикс доступа к URL HTTP

\*/

**public** **static** **final** String ***URL\_PREFIX\_HTTP*** = "http://";

/\*\*

\* Шаблон регулярных выражений, необходиый для поиска префиксов http://

\*/

**public** **static** **final** String ***PREFIX\_PATTERN*** = "(http://)";

/\*\*

\* Шаблон регулярных выражения соответствующий доменным именам

\*/

**public** **static** **final** String ***DOMAIN\_PATTERN*** = "([\\da-zа-я\\.-]+)\\.([a-zа-я\\.]{2,6})";

/\*\*

\* Шаблон регулярных выражений, соответсвующий путям до файла

\*/

**public** **static** **final** String ***PATH\_PATTERN*** = "([/\\wа-я\\.-]\*)\*\\/?";

/\*\*

\* Шаблон регулярных выражений, возвращающий расширение файла

\*/

**public** **static** **final** String ***EXTENSION\_PATTERN*** = "\\.[\\wа-я\\-]+$";

/\*\*

\* Шаблон регулярных выражений, соответствующий url строке

\*/

**public** **static** **final** String ***URL\_PATTERN*** = ***PREFIX\_PATTERN***+***DOMAIN\_PATTERN***+***PATH\_PATTERN***;

/\*\*

\* Скомплированный шаблон допустимых расширений

\*/

**public** **final** Pattern ACCEPTABLE\_EXTENSION\_PATTERN\_COMPILE = Pattern.*compile*(***ACCEPTABLE\_EXTENSION\_PATTERN***);

/\*\*

\* Скомпилированный шаблон регулярных выражений, необходиый для поиска префиксов http://

\*/

**public** **final** Pattern PREFIX\_PATTERN\_COMPILE = Pattern.*compile*(***PREFIX\_PATTERN***);

/\*\*

\* Скомпилированный шаблон регулярных выражения соответствующий доменным именам

\*/

**public** **final** Pattern DOMAIN\_PATTERN\_COMPILE = Pattern.*compile*(***DOMAIN\_PATTERN***);

/\*\*

\* Скомпилированный шаблон регулярных выражений, соответсвующий путям до файла

\*/

**public** **final** Pattern PATH\_PATTERN\_COMPILE = Pattern.*compile*(***PATH\_PATTERN***);

/\*\*

\* Скомпилированный шаблон регулярных выражений, возвращающий расширение файла

\*/

**public** **final** Pattern EXTENSION\_PATTERN\_COMPILE = Pattern.*compile*(***EXTENSION\_PATTERN***);

/\*\*

\* Скомпилированный шаблон регулярных выражений, соответствующий url строке

\*/

**public** **final** Pattern URL\_PATTERN\_COMPILE = Pattern.*compile*( ***URL\_PATTERN***);

/\*\*

\* Переменная содержащая url

\*/

**public** String url;

/\*\*

\* Переменная содержащая префикс доступа

\*/

**public** String prefix;

/\*\*

\* Переменная содержащая доменное имя

\*/

**public** String domain;

/\*\*

\* Переменная содержащая путь

\*/

**public** String path;

/\*\*

\* Переменная содержащая глубину

\*/

**public** **int** depth;

/\*\*

\* Конструктор для строки содержащей url

\* **@param** url - Входящая строка, содержащая url

\* **@param** depth - Глубина

\* **@throws** MalformedURLException - данная строка не содержит url

\*/

**public** URLDepthPair(String url, **int** depth) **throws** MalformedURLException

{

**if** (URL\_PATTERN\_COMPILE.matcher(url).matches()) **this**.url= url;

**else** **this**.url = searchPattern(url,URL\_PATTERN\_COMPILE);

Pattern p = **this**.DOMAIN\_PATTERN\_COMPILE;

Matcher m = p.matcher(**this**.url);

m.find();

**this**.domain = **this**.url.substring(m.start(),m.end());

**this**.path = **this**.url.substring(m.end());

**if** (**this**.path.isEmpty()) **this**.path = "/";

**this**.depth = depth;

checkExtension();

}

/\*\*

\* Конструктор для строки типа 'href="URL" '(С любым символом на конце после кавычек)

\* **@param** href - Передаваемая строка

\* **@param** depth - Глубина поиска, нак которой она найдена

\* **@param** domain - Домен, на котором найдена найдена данная строка

\* **@throws** MalformedURLException - В случае, если строка не содержит прямой или косвенной ссылки

\*/

**public** URLDepthPair(String href, **int** depth, String domain) **throws** MalformedURLException

{

String hrefContent = href.substring(***START\_HREF***,href.length()-***HREF\_TOEND***);

// Если строка содержит полную URL ссылку

**try**

{

**this**.url = searchPattern(hrefContent,URL\_PATTERN\_COMPILE);

Pattern p = **this**.DOMAIN\_PATTERN\_COMPILE;

Matcher m = p.matcher(**this**.url);

m.find();

**this**.domain = **this**.url.substring(m.start(),m.end());

**this**.path = **this**.url.substring(m.end());

**if** (**this**.path.isEmpty()) **this**.path = "/";

**this**.depth = depth;

checkExtension();

}

// Если строка имеет тип пути

**catch** (MalformedURLException e)

{

// Проверка правильности домена

**if** (!DOMAIN\_PATTERN\_COMPILE.matcher(domain).matches()) **throw** **new** MalformedURLException("Wrong URL");

hrefContent = searchPattern(hrefContent,PATH\_PATTERN\_COMPILE);

**if** (hrefContent.isEmpty()) **throw** **new** MalformedURLException("Wrong URL");

**if** (hrefContent.charAt(0) != '/') hrefContent = "/"+hrefContent;

**this**.url = ***URL\_PREFIX\_HTTP***+domain+hrefContent;

**this**.domain = domain;

**this**.path = hrefContent;

**this**.depth = depth;

checkExtension();

}

}

/\*\*

\* Метод поиска подстроки соответствующей шаблону

\* **@param** url - входная строка

\* **@param** pattern - искомый шаблон

\* **@return** String - найденная подстрока, содержащая url, иначе null

\* **@throws** MalformedURLException - Данная строка не содержит url

\*/

**public** String searchPattern(String url,Pattern p) **throws** MalformedURLException

{

Matcher m = p.matcher(url);

**try**

{

m.find();

**return** url.substring(m.start(),m.end());

}

**catch** (IllegalStateException e)

{

**throw** **new** MalformedURLException("Wrong URL");

}

}

/\*\*

\* Метод проверки на то, что расширение файла на конце найденного url

\* или отсутствует или является допустимым, в противном случае выдает ошибку

\* **@throws** MalformedURLException

\*/

**private** **void** checkExtension() **throws** MalformedURLException

{

String extension = **null**;

**try** {

extension = searchPattern(path,EXTENSION\_PATTERN\_COMPILE);

}

**catch** (MalformedURLException e)

{

extension = ***HTML\_EXTENSION***;

}

**finally**

{

**if** (!ACCEPTABLE\_EXTENSION\_PATTERN\_COMPILE.matcher(extension).matches()) **throw** **new** MalformedURLException("Wrong extension");

}

}

@Override

**public** **int** hashCode()

{

**return** url.hashCode();

}

@Override

**public** **boolean** equals(Object obj)

{

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

URLDepthPair other = (URLDepthPair) obj;

**if** (url.equals(other.url)) **return** **true**;

**else** **return** **false**;

}

}

/\*\*

\* Класс запускающий сканер Веб-Страниц

\*

\*/

**public** **class** Web\_Scan {

/\*\*

\* **@param** url - Адрес веб-страницы с которой начинается поиск

\* **@param** depth - Максимальная глубина поиска

\* **@throws** IOException

\* **@throws** UnknownHostException

\* **@throws** SQLException

\* **@throws** ClassNotFoundException

\*/

**public** **static** **void** main(String[] args) **throws** UnknownHostException, IOException, ClassNotFoundException, SQLException

{

//http://www.777gid.ru/p/blog-page\_30.html

String url = args[0];

**int** depth = Integer.*parseInt*(args[1]);

Crawler crawl = **new** Crawler(url, depth);

crawl.startMultiThreadCrawl(5);

}

}

/\*\*

\* Класс осуществляющий работу с http запросами с помощью сокета

\*/

**public** **class** SocketConnection

{

/\*\*

\* Код ответа сервера

\*/

**public** String code;

/\*\*

\* Используемый при соеденении протокол

\*/

**private** **final** String PROTOCOL = "HTTP/1.1";

/\*\*

\* Время ожидания ответа от сервара в мс.

\*/

**private** **final** **int** READ\_TIMEOUT = 20000;

/\*\*

\* Порт по которому идет соединение

\*/

**private** **final** **int** CONNECTION\_PORT = 80;

/\*\*

\* PrintWriter, через который отправляются данные на сервер

\*/

**public** PrintWriter pw;

/\*\*

\* BufferedReader, в которой осуществляется чтение ответа сервера

\*/

**public** BufferedReader br;

/\*\*

\* domain - домен, к которому ведется подключение

\*/

**public** String domain;

/\*\*

\* Путь до файла, без доменного имени

\*/

**public** String path;

/\*\*

\* Сокет с которым ведется связь

\*/

**private** Socket sock;

/\*\*

\* Содержит заголовки ответа на http запрос

\*/

**public** HashMap<String, String> request;

/\*\*

\* Конструктор работающий с объектом URLDepthPair

\* **@param** url - Объект URLDepthPair, содержащий данные о подключения к сайту

\*/

**public** SocketConnection(URLDepthPair url)

{

**this**.domain = url.domain;

**this**.path = url.path;

}

/\*\*

\* **@param** domain - Доменное имя

\* **@param** path - Путь до файла без доменного имени

\*/

**public** SocketConnection(String domain, String path)

{

**this**.domain = domain;

**this**.path = path;

}

/\*\*

\* **@param** domain - Доменное имя

\*/

**public** SocketConnection(String domain)

{

**this**.domain = domain;

**this**.path = "/";

}

/\*\*

\* Метод подключения к серверу

\* **@return** true, при удачном подключении

\*/

**public** **boolean** connect()

{

**try**

{

**this**.sock = **new** Socket(domain,CONNECTION\_PORT);

sock.setSoTimeout(READ\_TIMEOUT);

OutputStream os = sock.getOutputStream();

**this**.pw = **new** PrintWriter(os,**true**);

InputStream is = sock.getInputStream();

InputStreamReader in = **new** InputStreamReader(is);

**this**.br = **new** BufferedReader(in);

**return** **true**;

}

**catch** (UnknownHostException e)

{

**return** **false**;

}

**catch** (IOException e)

{

**return** **false**;

}

}

/\*\*

\* Метод отправляющий GET запрос на сервер

\* **@throws** IOException

\*/

**public** **void** sendGET () **throws** IOException

{

pw.println("GET "+path+" " + PROTOCOL);

pw.println("Host: "+ domain);

pw.println("");

getCode();

}

/\*\*

\* Метод, получающий код ответа

\* **@throws** IOException

\*/

**private** **void** getCode() **throws** IOException

{

**try**

{

String line = br.readLine();

code = line.substring(PROTOCOL.length()+1,PROTOCOL.length()+4);

// "Пролистываем не нужные html заголовки"

**while** (!(line = br.readLine()).equals(""));

}

**catch**(java.net.SocketTimeoutException e)

{

code = "ReadTimedOut";

}

}

/\*\*

\* Метод закрывающий соединение с сокетом

\* **@throws** IOException

\*/

**public** **void** close() **throws** IOException

{

**this**.sock.close();

}

}

/\*\*

\* Класс собирающий информацию о URL странице

\*/

**public** **class** InfoURL {

/\*\*

\* url данной страницы

\*/

**private** String url;

/\*\*

\* Содержит количество повторений слов длин больше 2

\*/

**private** HashMap<String,Integer> wordMapCount = **new** HashMap<String,Integer>();

/\*\*

\* Количество слов длины больше 2

\*/

**private** **int** wordCount = 0;

/\*\*

\* Количество ссылок на данную страниц из других источников

\*/

**private** **int** quoteCount = 1;

/\*\*

\* Количество ссылок на url внутри страницы

\*/

**private** **int** hrefCount = 0;

/\*\*

\* Количество заголовков на странице

\*/

**private** **int** hCount = 0;

/\*\*

\* "Индекс повторяемости" - индекс косвенно указывающий на неповторяемость текста

\*/

**private** **double** repeatIndex = 0;

/\*\*

\* "Индекс ссылок" - Индекс косвенно указывающий на количество ссылок на другие источники

\* внутра страницы

\*/

**private** **double** hrefIndex = 0;

/\*\*

\* "Индекс заголовков" - Индекс косвенно указывающий на количество заголовков

\*/

**private** **double** hIndex = 0;

/\*\*

\* "Индекс контента" - Индекс косвенно указывающий на качество содержания контента на данной странице

\*/

**private** **double** contentIndex = 0;

/\*\*

\* Заголвок данной страницы

\*/

**private** String title = "";

/\*\*

\* Описание данной страницы

\*/

**private** String description = "";

/\*\*

\* Флаг, указывающий что описание страницы еще не найдено

\*/

**private** **boolean** readDescription = **true**;

/\*\*

\* Флаг, указывающий что начало данной строки находится внутри метатега описания

\*/

**private** **boolean** isReadDescription = **false**;

/\*\*

\* Флаг, указывающий что начало данной строки находится внутри тега <style>

\*/

**private** **boolean** isReadStyle = **false**;

/\*\*

\* Флаг, указывающий что начало строки находится внутри тега script

\*/

**private** **boolean** isReadScript = **false**;

/\*\*

\* Флаг, укаывающий что заголовок страницы еще не найден

\*/

**private** **boolean** readTitle = **true**;

/\*\*

\* Максимальная повторяемость слов, чтобы попасть в список неповторяемых

\*/

**private** **final** **int** REPEAT\_COUNT= 5;

/\*\*

\* Коэффициент важности слов с низкой повторяемостью

\*/

**private** **final** **int** REPEAT\_IMPORTANT= 1000;

/\*\*

\* Коэффициент важности ссылок для общего индекса

\*/

**private** **final** **int** HREF\_IMPORTANT= 10000;

/\*\*

\* Коэффициент важности заголовков для общего индекса

\*/

**private** **final** **int** H\_IMPORTANT = 50000;

/\*\*

\* Отступ в начале для получения содержимого title

\*/

**private** **final** **int** START\_TITLE = 7;

/\*\*

\* Отступ в конце для получения содержимого title

\*/

**private** **final** **int** TOEND\_TITLE = 8;

/\*\*

\* Шаблон регулярных выражений для поиска мета тега с описанием

\*/

**public** **final** String DESCPRITION\_PATTERN = "(?i)(<meta.\*?name=\"description\".\*?content=\")";

/\*\*

\* Шаблон регулярных выражений для поиска мета тега с заголовком

\*/

**public** **final** String TITLE\_PATTERN = "(?i)(<title>).\*?(</title>)";

/\*\*

\* Шаблон регулярных выражений для поиска тегов, без их содержимого и тегов

\* script с их содержимым

\*/

**public** **final** String UNTAG\_PATTERN = "(?i)(<script.\*?>.\*?</script>)|(<(?!script).\*?>)";

/\*\*

\* Шаблон регулярных выражений для поиска открывающего тега script

\*/

**public** **final** String SCRIPT\_OPEN\_PATTERN = "(?i)<script.\*?>.\*";

/\*\*

\* Шаблон регулярных выражения для поиска закрывающего тега sciprt

\*/

**public** **final** String SCRIPT\_CLOSE\_PATTERN = "(?i)</script>";

/\*\*

\* Шаблон регулярных выражений для поиска открывающего тега style

\*/

**public** **final** String STYLE\_OPEN\_PATTERN = "(?i)<style.\*?>";

/\*\*

\* Шаблон регулярных выражений для поиска тега стайл с содержимым

\*/

**public** **final** String STYLE\_PATTERN = "(?i)<style.\*?>.\*?</style>";

/\*\*

\* Шаблон регулярных выражения для поиска закрывающего тега style

\*/

**public** **final** String STYLE\_CLOSE\_PATTERN = "(?i)</style>";

/\*\*

\* Шаблон регулярных выражений для поиска заголовков

\*/

**public** **final** String H\_TAG\_PATTERN = "(?i)<h[1-6]>";

/\*\*

\* Шаблон регулярных выражения для поиска слов

\*/

**public** **final** String WORD\_PATTERN = "[^а-яА-Яa-zA-Z]";

/\*\*

\* Скомпилированный шаблон для поиска мета тега с описанием

\*/

**private** **final** Pattern DESCPRITION\_PATTERN\_COMPILE = Pattern.*compile*(DESCPRITION\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска мета тега с заголовком

\*/

**private** **final** Pattern TITLE\_PATTERN\_COMPILE = Pattern.*compile*(TITLE\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска тегов, без их содержимого и тегов

\* script с их содержимым

\*/

**private** **final** Pattern UNTAG\_PATTERN\_COMPILE = Pattern.*compile*(UNTAG\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска открывающего тега script

\*/

**private** **final** Pattern SCRIPT\_CLOSE\_PATTERN\_COMPILE = Pattern.*compile*(SCRIPT\_CLOSE\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска закрывающего тега script

\*/

**private** **final** Pattern SCRIPT\_OPEN\_PATTERN\_COMPILE = Pattern.*compile*(SCRIPT\_OPEN\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска открывающего тега style

\*/

**private** **final** Pattern STYLE\_CLOSE\_PATTERN\_COMPILE = Pattern.*compile*(STYLE\_CLOSE\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска закрывающего тега style

\*/

**private** **final** Pattern STYLE\_OPEN\_PATTERN\_COMPILE = Pattern.*compile*(STYLE\_OPEN\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска тега style с содержимым

\*/

**private** **final** Pattern STYLE\_PATTERN\_COMPILE = Pattern.*compile*(STYLE\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска заголовков

\*/

**private** **final** Pattern H\_TAG\_PATTERN\_COMPILE = Pattern.*compile*(H\_TAG\_PATTERN);

/\*\*

\* Скомпилированный шаблон для поиска слов

\*/

**private** **final** Pattern WORD\_PATTERN\_COMPILE = Pattern.*compile*(WORD\_PATTERN);

/\*\*

\* Конструктор класса

\* **@param** url - url данного сайта

\*/

**public** InfoURL(String url)

{

**this**.url = url;

}

/\*\*

\* Метод добавляющий количество "цитирования" данной страницы другими

\*/

**public** **void** addQuoute()

{

quoteCount+=1;

}

/\*\*

\* Метод добавляющий количество ссылок на url на данной странице

\*/

**public** **void** addHref()

{

hrefCount+=1;

}

/\*\*

\* Метод занимающийся поиском описания страницы

\* **@param** line - передаваемая строка страницы

\*/

**private** **void** readDescription(String line)

{

**if** (**this**.readDescription)

{

Matcher m = **this**.DESCPRITION\_PATTERN\_COMPILE.matcher(line);

**try**

{

m.find();

line = line.substring(m.end());

**this**.isReadDescription = **true**;

**this**.readDescription = **false**;

}

**catch** (IllegalStateException e)

{

}

}

**if** (isReadDescription)

{

**int** index = line.indexOf("\"");

**if** (index == -1)

{

**this**.description+=line;

}

**else**

{

**this**.description += line.substring(0, index);

**this**.isReadDescription = **false**;

}

}

}

/\*\*

\* Метод занимающийся поиском заголовка title страницы

\* **@param** line - передаваемая строка страницы

\*/

**private** **void** readTitle(String line)

{

**if** (**this**.readTitle)

{

Matcher m = **this**.TITLE\_PATTERN\_COMPILE.matcher(line);

**try**

{

m.find();

**this**.title = line.substring(m.start()+**this**.START\_TITLE,m.end()-**this**.TOEND\_TITLE);

**this**.readTitle = **false**;

}

**catch** (IllegalStateException e)

{

}

}

}

/\*\*

\* Метод занимающийся поиском заголовков на странице

\* **@param** line - передаваемая строка страницы

\*/

**private** **void** parseH(String line)

{

Matcher m = **this**.H\_TAG\_PATTERN\_COMPILE.matcher(line);

**while**(m.find())

{

**this**.hCount++;

}

}

/\*\*

\* Метод занимающийся поиском повторяемости слов

\* **@param** line - передаваемая строка страницы

\*/

**private** **void** parseWord(String line)

{

**if** (**this**.isReadScript)

{

Matcher m = **this**.SCRIPT\_CLOSE\_PATTERN\_COMPILE.matcher(line);;

**try**

{

m.find();

line = line.substring(m.end());

**this**.isReadScript = **false**;

}

**catch** (IllegalStateException e)

{

}

}

**if**(!**this**.isReadScript)

{

Matcher m = **this**.UNTAG\_PATTERN\_COMPILE.matcher(line);

String parseLine = m.replaceAll(" ");

m = **this**.SCRIPT\_OPEN\_PATTERN\_COMPILE.matcher(parseLine);

**if** (m.find())

{

parseLine = m.replaceFirst("");

**this**.isReadScript = **true**;

}

m = **this**.WORD\_PATTERN\_COMPILE.matcher(parseLine);

parseLine = m.replaceAll(" ");

**for** (String var : parseLine.split(" "))

{

**if**(var.length() > 2)

{

**if**(wordMapCount.get(var)!=**null**) wordMapCount.put(var, wordMapCount.get(var)+1);

**else** wordMapCount.put(var, 1);

wordCount++;

}

}

}

}

/\*\*

\* Метод для удаления тега style из строки или игнорирования данной строки

\* если тег на ней не заканчивается

\* **@param** line - передаваемая строка

\* **@return** новая строка без тега style

\*/

**private** String readStyle(String line)

{

**if** (**this**.isReadStyle)

{

Matcher m = **this**.STYLE\_CLOSE\_PATTERN\_COMPILE.matcher(line);

**try**

{

m.find();

line = line.substring(m.end());

**this**.isReadStyle = **false**;

}

**catch** (IllegalStateException e)

{

}

}

**if**(!**this**.isReadScript)

{

Matcher m = **this**.STYLE\_PATTERN\_COMPILE.matcher(line);

String parseLine = m.replaceAll(" ");

m = **this**.STYLE\_OPEN\_PATTERN\_COMPILE.matcher(parseLine);

**if** (m.find())

{

parseLine = m.replaceFirst("");

**this**.isReadScript = **true**;

}

}

**return** line;

}

/\*\*

\* Метод рассчитывающий индексы содержания страницы

\*/

**private** **void** calculateIndex()

{

**if**(wordCount > 0)

{

**int** sum = 0;

**for** (Integer var : wordMapCount.values())

{

**if**(var<**this**.REPEAT\_COUNT)

{

sum+=var;

}

}

**if**(sum>0)

{

repeatIndex = **this**.REPEAT\_IMPORTANT\*sum/**this**.wordCount;

}

hrefIndex = **this**.HREF\_IMPORTANT\***this**.hrefCount/**this**.wordCount;

hIndex = **this**.H\_IMPORTANT\***this**.hCount/**this**.wordCount;

contentIndex = repeatIndex+hrefIndex+hIndex;

}

}

/\*\*

\* Метод читающий передаваемую строку

\* **@param** line - передаваемая строка страницы

\*/

**public** **void** readLine(String line)

{

line = readStyle(line);

**if**(!**this**.isReadStyle)

{

readDescription(line);

readTitle(line);

parseH(line);

parseWord(line);

calculateIndex();

}

}

/\*\*

\* **@return** Индекс повторяемости

\*/

**public** **double** getRepeatIndex()

{

**return** **this**.repeatIndex;

}

/\*\*

\* **@return** Индекс ссылок

\*/

**public** **double** getHrefIndex()

{

**return** **this**.hrefIndex;

}

/\*\*

\* **@return** Индекс заголовков

\*/

**public** **double** getHIndex()

{

**return** **this**.hIndex;

}

/\*\*

\* **@return** Индекс содержимого страницы

\*/

**public** **double** getContentIndex()

{

**return** **this**.contentIndex;

}

/\*\*

\* **@return** Количество "цитирования" страницы

\*/

**public** **int** getQuoteCount()

{

**return** **this**.quoteCount;

}

/\*\*

\* **@return** Заголовок страницы

\*/

**public** String getTitle()

{

**return** **this**.title;

}

/\*\*

\* **@return** Описание страницы

\*/

**public** String getDescription()

{

**return** **this**.description;

}

/\*\*

\* **@return** url страницы

\*/

**public** String getURL()

{

**return** **this**.url;

}

}

/\*\*

\* Класс для записи полученных URL и информации о них в базу данных

\*/

**public** **class** DBWriteURL {

/\*\*

\* Экзлемпляр объекта Connection базы данных к которой ведется подключение

\*/

**private** Connection conn;

/\*\*

\* Экзлемпляр объекта Statement базы данных к которой ведется подключение

\*/

**private** Statement statmt;

/\*\*

\* Используемый драйвер JDBC для подключения к базе данных

\*/

**private** **static** **final** String ***DRIVER*** = "org.sqlite.JDBC";

/\*\*

\* Префикс драйвера в пути для подключения к базе данных

\*/

**private** **static** **final** String ***DRIVER\_PREFIX*** = "jdbc:sqlite:";

/\*\*

\* Путь к файлу БД

\*/

**private** **static** **final** String ***PATH*** = "db\\";

/\*\*

\* Имя БД, к которой ведется подключение

\*/

**private** **static** **final** String ***DB*** = "parseurl.db";

/\*\*

\* Полный путь к БД для подключения

\*/

**private** **static** **final** String ***FULLPATH*** = ***DRIVER\_PREFIX***+***PATH***+***DB***;

/\*\*

\* Метод осуществляющий подключение к БД

\* **@throws** ClassNotFoundException

\* **@throws** SQLException

\*/

**public** **void** conn()

{

conn = **null**;

**try** {

Class.*forName*(***DRIVER***);

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

conn = DriverManager.*getConnection*(***FULLPATH***);

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Метод осуществляющий создание таблицы

\* **@throws** ClassNotFoundException

\* **@throws** SQLException

\*/

**public** **void** createDB()

{

**try** {

statmt = conn.createStatement();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

statmt.executeUpdate("CREATE TABLE if not exists 'url' "

+ "('id' INTEGER PRIMARY KEY AUTOINCREMENT,"

+ "'url' text,"

+ "'quouteCount' INT,"

+ "'contentIndex' INT,"

+"'title' text,"

+ "'desciption' text);");

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Создание в базе данных записи о проверенной странице

\* **@param** info - класс, содержащий информацию о сайте на данном url

\* **@throws** SQLException

\*/

**public** **void** writeDB(InfoURL info)

{

**try** {

statmt.executeUpdate("INSERT INTO 'url'"

+ " ('url', 'quouteCount','contentIndex','title','desciption') VALUES "

+ "('"+info.getURL().replaceAll("'", "\"")+"',"+info.getQuoteCount()+","+info.getContentIndex()+",'"+info.getTitle().replaceAll("'", "\"")+"','"+info.getDescription().replaceAll("'", "\"")+"');");

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Обновление количества "Цитирования" данной страницы другими

\* **@param** info - класс, содержащий информацию о сайте на данном url

\* **@throws** SQLException

\*/

**public** **void** updateQuoteDB(InfoURL info)

{

**try** {

statmt.executeUpdate("UPDATE 'url' "

+ "SET 'quouteCount' = "+info.getQuoteCount()

+ " WHERE 'url' = '"+info.getURL()+"'");

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* Метод закрывающий подключение к БД

\* **@throws** ClassNotFoundException

\* **@throws** SQLException

\*/

**public** **void** closeDB()

{

**try** {

conn.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**try** {

statmt.close();

} **catch** (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

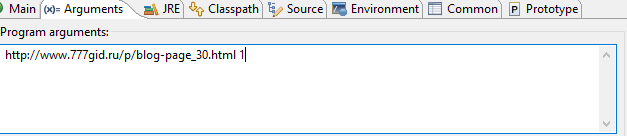
}

}

}

# Результат работы программы

Введенные как входные данные аргументы

 Некоторая часть вывода просмотренных сайтов



Некоторая часть записей в базу данных

