# Содержание:

[**Содержание:**](#_ssuts2nnjqda) **1**

[**Логика работы программы:**](#_g5vyzg22n44p) **2**

[**Теория:**](#_df28ugqalzi9) **2**

[Потоки ввода\вывода в java:](#_n8vfho71bmp8) 2

[Классы позволяющие ускорить чтение/запись за счет использования буфера:](#_8n4ksevcnx5o) 5

[Url, HTTP:](#_rvrx5waiw9vi) 6

[Сокеты:](#_51jkv8a3l3a7) 9

[LinkedList:](#_rnjam65u6rk4) 11

[**Возможные теоретические вопросы:**](#_1h15y0cmzwtn) **12**

[**Ссылки:**](#_q1mn9zw3o1h7) **12**

# Логика работы программы:

Принцип работы программы:

1. получение следующей ссылки из списка необработанных ссылок
2. создание подключения
3. если успешно:

3.1) создание get запроса для получения html кода страницы

3.2) поиск ссылок в полученном html коде

3.3) добавление найденных ссылок в список необработанных ссылок

1. закрытие сокета
2. если максимальная глубина поиска достигнута - завершение выполнения программы, иначе переход к шагу 1

# Теория:

## Потоки ввода\вывода в java:

В данном случае применительно к работе с вводом\выводом, поток (stream) следует понимать, как абстракцию, которая используется для чтения или записи информации. Объект, из которого можно считать данные, называется **потоком ввода**, а объект, в который можно записывать данные, - **потоком вывода**. Например, если надо считать из консоли, то применяется поток ввода, а если надо вывести в консоль - то поток вывода.

Java IO (input-output) является потокоориентированным. Потокоориентированный ввод/вывод подразумевает чтение/запись из потока/в поток одного или нескольких байт в единицу времени поочередно. Данная информация нигде не кэшируются. Таким образом, невозможно произвольно двигаться по потоку данных вперед или назад. Потоки ввода/вывода в Java IO являются блокирующими. Это значит, что когда в потоке выполнения вызывается read() или write() метод любого класса из пакета java.io.\*, происходит блокировка до тех пор, пока данные не будут считаны или записаны. Поток выполнения в данный момент не может делать ничего другого.

Каналы (channels) – это логические (не физические) порталы, абстракции объектов более низкого уровня файловой системы (например, отображенные в памяти файлы и блокировки файлов), через которые осуществляется ввод/вывод данных, а буферы являются источниками или приёмниками этих переданных данных. При организации вывода, данные, которые необходимо отправить, помещаются в буфер, который затем передается в канал. При вводе, данные из канала помещаются в заранее предоставленный буфер.

Каналы напоминают трубопроводы, по которым эффективно транспортируются данные между буферами байтов и сущностями по ту сторону каналов. Каналы – это шлюзы, которые позволяют получить доступ к сервисам ввода/вывода операционной системы с минимальными накладными расходами, а буферы – внутренние конечные точки этих шлюзов, используемые для передачи и приема данных.

Разделяют два вида потоков ввода/вывода:

* Байтовые:
  + *InputStream* и его наследники - совокупность для получения байтовых данных из различных источников;
  + *OutputStream* и его наследники - набор классов, определяющих потоковый байтовый вывод;
* Символьные:
  + *Reader* и его наследники определяют потоковый ввод символов Unicode;
  + *Writer* и его наследники определяют потоковый вывод символов Unicode.

Все остальные классы для работы с вводом\выводом чего-либо наследуются от этих четырех абстрактных классов.

Примеры классов наследников:

InputStream:

* *ByteArrayInputStream позволяет использовать буфер в памяти (массив байтов) в качестве источника данных для входного потока;*
* *DataInputStream - входной поток для байтовых данных, включающий методы для чтения стандартных типов данных Java;*
* *FileInputStream - входной поток для чтения информации из файла;*
* *ObjectInputStream - входной поток для объектов;*
* *StringBufferInputStream превращает строку (String) во входной поток данных InputStream;*
* *FilterInputStream - абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства;*
* *PipedInputStream реализует понятие входного канала;*
* *PushbackInputStream - разновидность буферизации, обеспечивающая чтение байта с последующим его возвратом в поток, позволяет «заглянуть» во входной поток и увидеть, что оттуда поступит в следующий момент, не извлекая информации.*
* *SequenceInputStream используется для слияния двух или более потоков InputStream в единый.*

OutputStream:

* *DataOutputStream - выходной поток байт, включающий методы для записи стандартных типов данных Java;*
* *FileOutputStream - запись данных в файл на физическом носителе;*
* *PrintStream - выходной поток, включающий методы print() и println();*
* *ObjectOutputStream - выходной поток для записи объектов;*
* *PipedOutputStream реализует понятие выходного канала.*
* *FilterOutputStream - абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства;*
* *ByteArrayOutputStream - все данные, посылаемые в этот поток, размещаются в предварительно созданном буфере;*
* Reader:
* *CharArrayReader - входной поток, который читает из символьного массива;*
* *FileReader - входной поток, читающий файл;*
* *InputStreamReader- входной поток, транслирующий байты в символы;*
* *LineNumberReader - входной поток, подсчитывающий строки;*
* *StringReader - входной поток, читающий из строки.*
* *PipedReader - входной канал;*
* *PushbackReader - входной поток, позволяющий возвращать символы обратно в поток;*
* *FilterReader - абстрактный класс, предоставляющий интерфейс для классов-надстроек;*

Writer:

* *CharArrayWriter - выходной поток, который пишет в символьный массив;*
* *FileWriter - выходной поток, пишущий в файл;*
* *PrintWriter - выходной поток символов, включающий методы print() и println();*
* *StringWriter - выходной поток, пишущий в строку;*
* *FilterWriter - абстрактный класс, предоставляющий интерфейс для классов-надстроек;*
* *OutputStreamWriter - выходной поток, транслирующий байты в символы;*
* *PipedWriter - выходной канал;*

## Классы позволяющие ускорить чтение/запись за счет использования буфера:

* BufferedInputStream(InputStream in)/BufferedInputStream(InputStream in, int size),
* BufferedOutputStream(OutputStream out)/BufferedOutputStream(OutputStream out, int size),
* BufferedReader(Reader r)/BufferedReader(Reader in, int sz),
* BufferedWriter(Writer out)/BufferedWriter(Writer out, int sz)

## Url, HTTP:

**URL**: унифицированный указатель ресурса. Это адрес веб страницы. Он имеет следующую структуру:

1) метод доступа к ресурсу;

2) доменное имя

3) путь к файлу

4) данные о файле

В данной лабораторной работе будет рассматривается метод доступа «http://».

**HTTP**: Hyper Text Transfer Protocol (Протокол передачи гипертекста). Это стандартный текстовый протокол, используемый для передачи данных веб-страницы через Интернет. HTTPS (HyperText Transfer Protocol Secure) является расширением протокола HTTP. HTTPS широко используется для защиты информации.

Структура протокола:

* Стартовая строка (starting line) — определяет тип сообщения;
* Заголовки (headers) — характеризуют тело сообщения, параметры передачи и прочие сведения;
* Тело сообщения (message body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Заголовки и тело сообщения могут отсутствовать, но стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа.

Методы отправки данных с клиента на сервер

* GET - используется для запроса содержимого указанного ресурса, изображения или гипертекстового документа. Вместе с запросом могут передаваться дополнительные параметры как часть URI, значения могут выбираться из полей формы или передаваться непосредственно через URL. При этом запросы кэшируются и имеют ограничения на размер. Этот метод является основным методом взаимодействия браузера клиента и веб-сервера.
* POST - используется для передачи пользовательских данных в содержимом HTTP-запроса на сервер. Пользовательские данные упакованы в тело запроса согласно полю заголовка Content-Type и/или включены в URI запроса. При использовании метода POST под URI подразумевается ресурс, который будет обрабатывать запрос.

Разница между методами GET и POST?

* GET передает данные серверу используя URL, тогда как POST передает данные, используя тело HTTP запроса. Длина URL ограничена 1024 символами, это и будет верхним ограничением для данных, которые можно отослать через GET. POST может отправлять гораздо большие объемы данных. Лимит устанавливается web-server и составляет обычно около 2 Mb.
* Передача данных методом POST более безопасна, чем методом GET, так как секретные данные (например пароль) не отображаются напрямую в web-клиенте пользователя, в отличии от URL, который виден почти всегда. Иногда это преимущество превращается в недостаток - вы не сможете послать данные за кого-то другого.
* GET метод является неизменяемым, тогда как POST — изменяемый.

Пример HTTP запроса и ответа сервера:

Запрос:

GET /index.php HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5

Accept: text/html

Connection: close

Первая строка — это строка запроса, остальные — заголовки; тело сообщения отсутствует

Ответ:

HTTP/1.0 200 OK

Server: nginx/0.6.31

Content-Language: ru

Content-Type: text/html; charset=utf-8

Content-Length: 1234

Connection: close

## Сокеты:

**Сокет**: Сокет(разъем) - это ресурс, предоставляемый операционной системой, который позволяет вам обмениваться данными с другими компьютерами по сети. Сокет используют для установки соединения с веб-сервером.

Класс *Socket* реализует идею сокета. Через его каналы ввода/вывода будут общаться клиент с сервером. Объявляется этот класс на стороне клиента, а сервер воссоздаёт его, получая сигнал на подключение. Так происходит общение в сети. Возможные конструкторы класса *Socket*:

* Socket(String имя\_хоста, int порт) throws UnknownHostException, IOException
* Socket(InetAddress IP-адрес, int порт) throws UnknownHostException

«имя\_хоста» — определённый узел сети, ip-адрес. Если класс сокета не смог преобразовать его в реальный, существующий, адрес, то сгенерируется исключение UnknownHostException. При потере соединения может произойти исключение IOException.

Самые часто используемые методы класса Socket:

* *InetAddress getInetAddress*() – возвращает объект содержащий данные о сокете. В случае если сокет не подключен – *null*
* *int getPort*() – возвращает порт по которому происходит соединение с сервером
* *int getLocalPort*() – возвращает порт к которому привязан сокет. Дело в том, что «общаться» клиент и сервер могут по одному порту, а порты, к которым они привязаны – могут быть совершенно другие
* *boolean isConnected*() – возвращает true, если соединение установлено
* *void connect*(SocketAddress адрес) – указывает новое соединение
* *boolean isClosed*() – возвращает true, если сокет закрыт
* *boolean isBound*() - возвращает true, если сокет действительно привязан к адресу

Класс *Socket* реализует интерфейс AutoCloseable, поэтому его можно использовать в конструкции try-with-resources. Закрыть сокет также можно классическим образом, с помощью *close*().

Пример создания сокета для обращения к time server:

import java.io.IOException;

import java.net.InetSocketAddress;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Scanner;

public class SocketExample {

public static void main(String[] args) throws IOException, UnknownHostException{

try(Socket socket = new Socket()){

socket.connect( new InetSocketAddress("129.6.15.28", 13), 2000);

Scanner scanner = new Scanner(socket.getInputStream());

while(scanner.hasNextLine()){

System.out.println(scanner.nextLine());

}

}

}

}

**Порт**: порт - уникальный номер, необходимый для установки соединения. несколько разных программ на одном сервере могут слушать соединения через разные порты. Каждый порт обозначается номером в диапазоне 1..65535. Номера от 1 до 1024 зарезервированы для операционной системы. У большинства серверов есть порт по умолчанию. Для HTTP соединений обычно используется порт 80.

## **LinkedList:**

В LinkedList элементы представляют собой звенья одной цепи. У каждого элемента помимо тех данных, которые он хранит, имеется ссылка на предыдущий и следующий элемент. По этим ссылкам можно переходить от одного элемента к другому.

**Вся работа с LinkedList сводится к изменению ссылок.**

Прежде всего, LinkedList эффективно использовать в работе с серединой списка. Вставка и удаление в середину LinkedList устроены гораздо проще, чем в ArrayList. Ссылки соседних элементов переопределяются , а ненужный элемент “выпадает” из цепочки ссылок.

Пример работы с LinkedList:

import java.util.LinkedList;

public class LinkedListExample {

public static void main(String[] args) {

String str1 = new String("Hello ");

String str2 = new String("world");

String str3 = new String("!");

LinkedList<String> list = new LinkedList<>();

list.add(str1);

list.add(str3);

list.add(1, str2);

System.out.println(list);

// будет выведено [Hello , world, !]

}

}

# 

# Возможные теоретические вопросы:

* что такое сокет?
* что такое порт?
* что такое URL?
* что такое HTTP протокол?
* как сформировать HTTP запрос?
* как установить соединение с сервером?
* как получить html код страницы сайта?
* каким образом происходит поиск ссылок на сайте в программе?
* Что такое LinkedList и для чего мы его используем в работе?

# Ссылки:

1. Формирование GET запроса для получения htlm кода страницы в java: <https://youtu.be/rzeA7tAKVCA>
2. Сокеты в java: <https://youtu.be/Jji5NPVmbAE>
3. Статья о LinkedList: <https://javarush.ru/groups/posts/1938-linkedlist>