**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение высшего**

**образование**

**Ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

**Кафедра «МКиИТ»**

**дисциплина «Объектно-ориентированное программирование»**

Отчет по Лабораторной работе №7

Подготовила студентка

группы БВТ1901: Нкурикийе Х

Проверил: Мосева М.С.

Москва 2021

**Crawler**

import java.io.\*;

import java.net.\*;

import java.util.\*;

import java.util.regex.\*;

import java.util.concurrent.LinkedBlockingDeque;

import java.sql.SQLException;

public class Crawler {

static final String CONST1 = "(href=\").\*?\"";

static final String CONST2 = "(<a).\*?>";

private static final Pattern TAG\_A\_PATTEN\_COMPILE = Pattern.compile(CONST2);

private static final Pattern CONST1\_COMPILE = Pattern.compile(CONST1);

private HashSet<String> enter = new HashSet<String>();

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

private LinkedBlockingDeque <URLDepthPair> unVisited = new LinkedBlockingDeque <URLDepthPair>();

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

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

private int depth;

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

String test = "http://pistiaistyoryhma.myspecies.info";

Crawler crawl = new Crawler(test, 1);

crawl.startCrawl();

}

public Crawler(String url, int depth) throws UnknownHostException, IOException, ClassNotFoundException, SQLException {

URLDepthPair urlpair = new URLDepthPair(url,0);

unVisited.add(urlpair);

enter.add(urlpair.url);

allDomain.add(urlpair.domain);

this.depth = depth;

}

public void startCrawl() throws IOException, InterruptedException {

while(!unVisited.isEmpty()) {

URLDepthPair urlpair = unVisited.pollFirst();

if (urlpair.depth<=depth)

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

else noneCheckedList.add(urlpair);

}

}

private boolean readURL(URLDepthPair urlpair) throws IOException {

try {

SocketConnection conn = new SocketConnection(urlpair);

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

conn.sendGET();

if (Integer.parseInt(conn.code) == 200) {

System.out.printf("[%s]: good request, status code: %s\n", urlpair.url, conn.code);

String line;

BufferedReader buffReader = conn.buffReader;

while ((line = buffReader.readLine()) != null) {

ParseNewURL(line,urlpair);

}

conn.close();

return true;

}

else {

System.out.printf("[%s]: bad request, status code: %s\n", urlpair.url, conn.code);

return false;

}

}

catch (SocketTimeoutException exception) {

System.out.printf("[%s]: connection timeout\n", urlpair.url);

return false;

}

}

private boolean ParseNewURL(String line, URLDepthPair url){

try {

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

m.find();

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

Matcher m2 = CONST1\_COMPILE.matcher(tagA);

m2.find();

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

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

if (!enter.contains(newURL.url)) {

enter.add(newURL.url);

if (allDomain.contains(newURL.domain)) unVisited.addLast(newURL);

else {

unVisited.addFirst(newURL);

allDomain.add(newURL.domain);

}

return true;

}

else return false;

}

catch (IllegalStateException e) {

return false;

}

catch (MalformedURLException e) {

return false;

}

}

public LinkedList <URLDepthPair> getcheckedSites() {

return checkedList;

}

public LinkedList <URLDepthPair> getUncheckedSites() {

return noneCheckedList;

}

}

**SocketConnection**

import java.io.\*;

import java.net.\*;

import java.util.HashMap;

public class SocketConnection {

public String code;

public String domain;

public String path;

private Socket sock;

private final String PROTOCOL = "HTTP/1.1";

private final int READ\_TIMEOUT = 20000;

private final int CONNECTION\_PORT = 80;

public PrintWriter pWriter;

public BufferedReader buffReader;

public HashMap<String, String> request;

public SocketConnection(URLDepthPair url) {

this.domain = url.domain;

this.path = url.path;

}

public SocketConnection(String domain, String path) {

this.domain = domain;

this.path = path;

}

public SocketConnection(String domain) {

this.domain = domain;

this.path = "/";

}

public boolean connect() {

try {

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

sock.setSoTimeout(READ\_TIMEOUT);

OutputStream os = sock.getOutputStream();

this.pWriter = new PrintWriter(os,true);

InputStream is = sock.getInputStream();

InputStreamReader in = new InputStreamReader(is);

this.buffReader = new BufferedReader(in);

return true;

}

catch (UnknownHostException e) {

return false;

}

catch (IOException e) {

return false;

}

}

public void sendGET () throws IOException {

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

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

pWriter.println("");

getCode();

}

private void getCode() throws IOException {

String line = buffReader.readLine();

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

while (!(line = buffReader.readLine()).equals(""));

}

public void close() throws IOException {

this.sock.close();

}

}

// **URLDepthPair**

import java.net.MalformedURLException;

import java.util.regex.\*;

public class URLDepthPair {

public static final int START\_HREF = 6;

public static final int HREF\_TOEND = 1;

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

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

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

public static final String CONST1 = "([\\da-zа-я\\.-]+)\\.([a-zа-я\\.]{2,6})";

public static final String CONST2 = "([/\\wа-я\\.-]\*)\*\\/?";

public static final String CONST3 = "\\.[\\wа-я\\-]+$";

public static final String URL\_PATTERN = PREFIX\_PATTERN+CONST1+CONST2;

public String url;

public String prefix;

public String domain;

public String path;

public int depth;

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

if (testURL(url)) this.url= url;

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

Pattern p = Pattern.compile(CONST1);

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();

}

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

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

try {

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

Pattern p = Pattern.compile(CONST1);

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();

}

catch (MalformedURLException e) {

if (!Pattern.matches(CONST1,domain)) throw new MalformedURLException("Wrong URL");

hrefContent = searchPattern(hrefContent,CONST2);

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();

}

}

public boolean testURL(String url) {

return Pattern.matches(URL\_PATTERN, url);

}

public String searchPattern(String url,String pattern) throws MalformedURLException {

Pattern p = Pattern.compile(pattern);

Matcher m = p.matcher(url);

try {

m.find();

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

}

catch (IllegalStateException e) {

throw new MalformedURLException("Wrong URL");

}

}

private void checkExtension() throws MalformedURLException {

String extension = null;

try {

extension = searchPattern(path,CONST3);

}

catch (MalformedURLException e) {

extension = HTML\_EXTENSION;

}

finally {

if (!extension.equals(HTML\_EXTENSION)) 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;

}

}

