## CS 3031: Telecommunications

Assignment #1: A Web Proxy Server

Jakub Slowinski: 16319781

Code at end of report + view my repository @ https://github.com/slow-J/CS3031-Telecoms

library used: tcdlib from CS2031

language used: java

Web proxy client class:

The client operates on port 2000

The client takes in an input to be the client number which makes it operate on port: 2000+client\_no.

Due to this multiple clients can operate simultaneously.

My client class takes in a string which becomes the payload of the message being send to the server.

The client\_no is stored in the header[0] of the packet the client sends, to know who sent it.

header[1] of the packet is initialised to -1 to show that the message hasnt passed through the blacklist yet(happens in management console).

The clients can only send packets to the proxy server.

The client prints messages received.

Proxy server class:

The gateway operates on port 4000.

It can send packets to the client or management console.

On receipt, it filters the message by buffer[1], which if -1 means it must be rerouted to management console for blacklist filtering, 0 if it’s been checked and its banned, 1 for an approved http request.

There is an LRU cache instantiated on running the server and checks cache before doing a GET request and if not in cache, adds to cache upon receiving response from website.

If banned, it sends a packet to client(client\_no from buffer[0]) which sent it, detailing that the website requested is banned.

If buffer[1]==-1, reroute packet to management console.

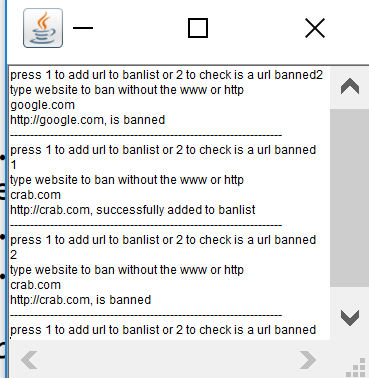
The server is threaded, allowing many clients to operate simultaneously.

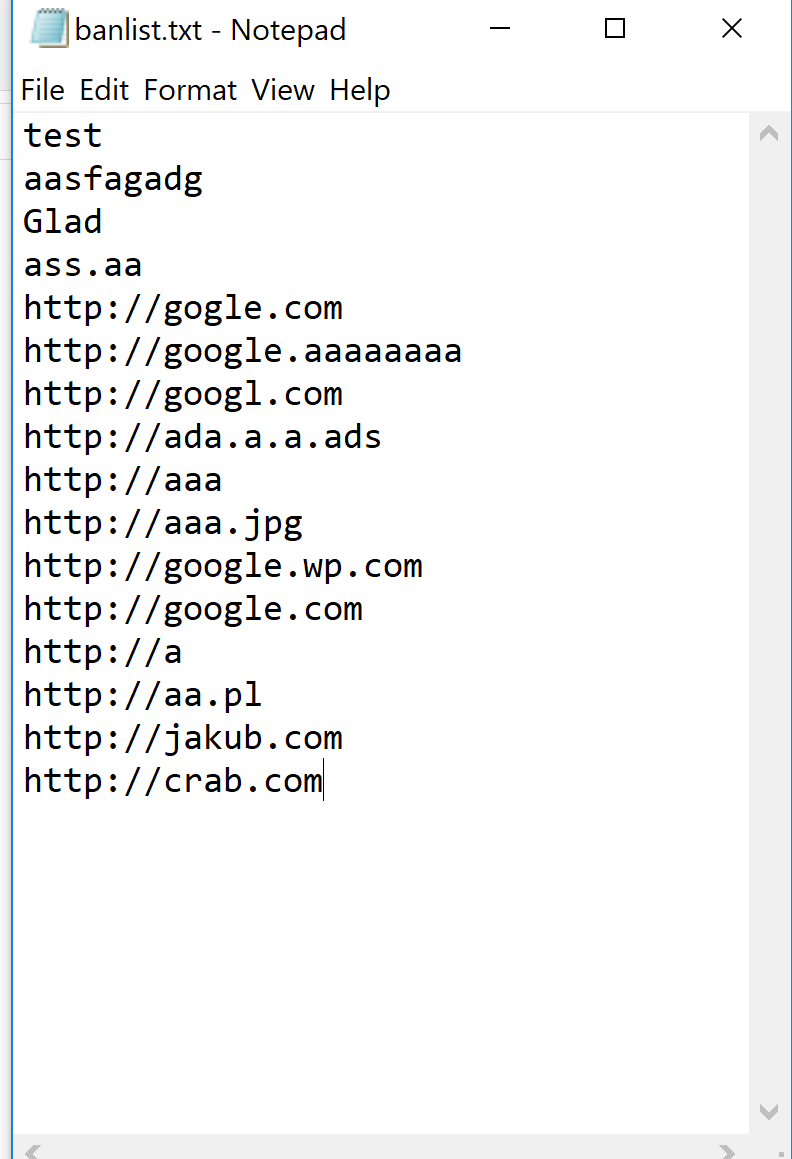
Management console class:

Operates on port 2000.

The main operation here is to implement a persistent ban list, which is stored as a txt file.

It can dynamically add websites to the ban list as well as check if a website is banned.

It displays each http request on the console.



LRUcache & LRUnode class:

Implements a least recently used cache to cache requests locally to save bandwidth.

You can set the capacity to any desired number. I had it set to 4 in my program.

The cache is used in the proxy server class.

Packet content class:

The packet content class serves as an interface. It possesses the toString() and toDatagramPacket() methods. It also holds the length of the header as HEADERLENGTH.

This interface along with string content and node classes are from my last years asssignments from CS2031

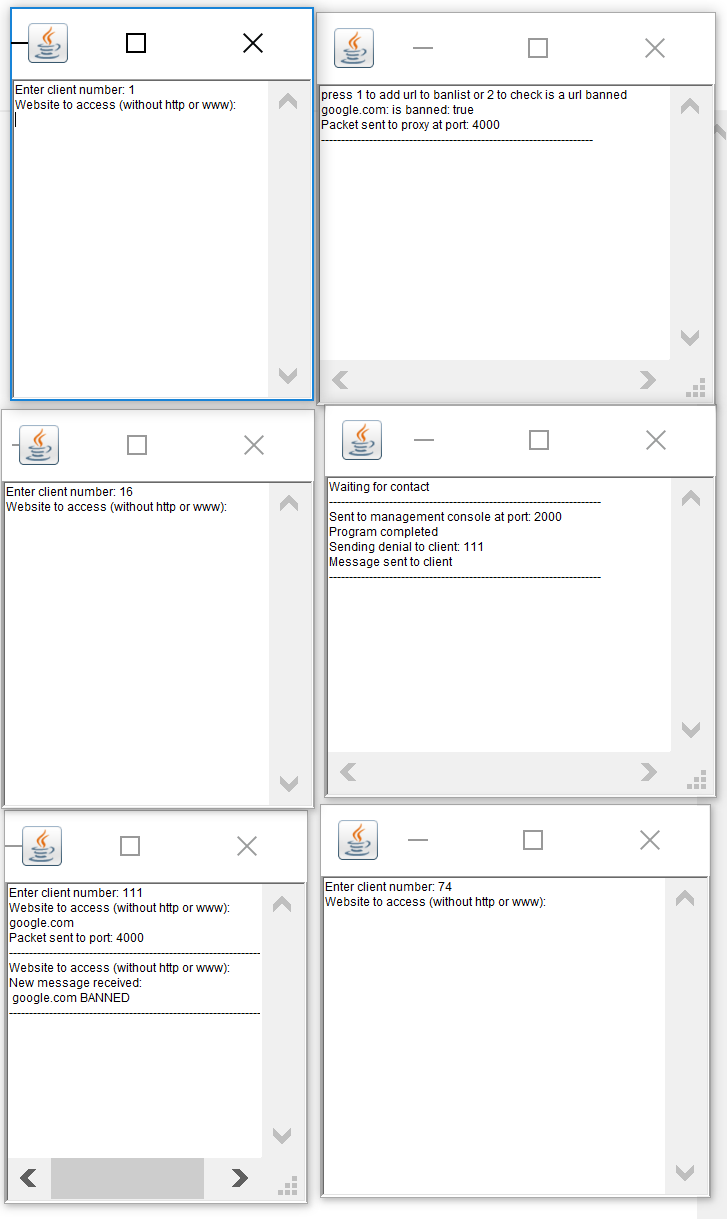
String content class:

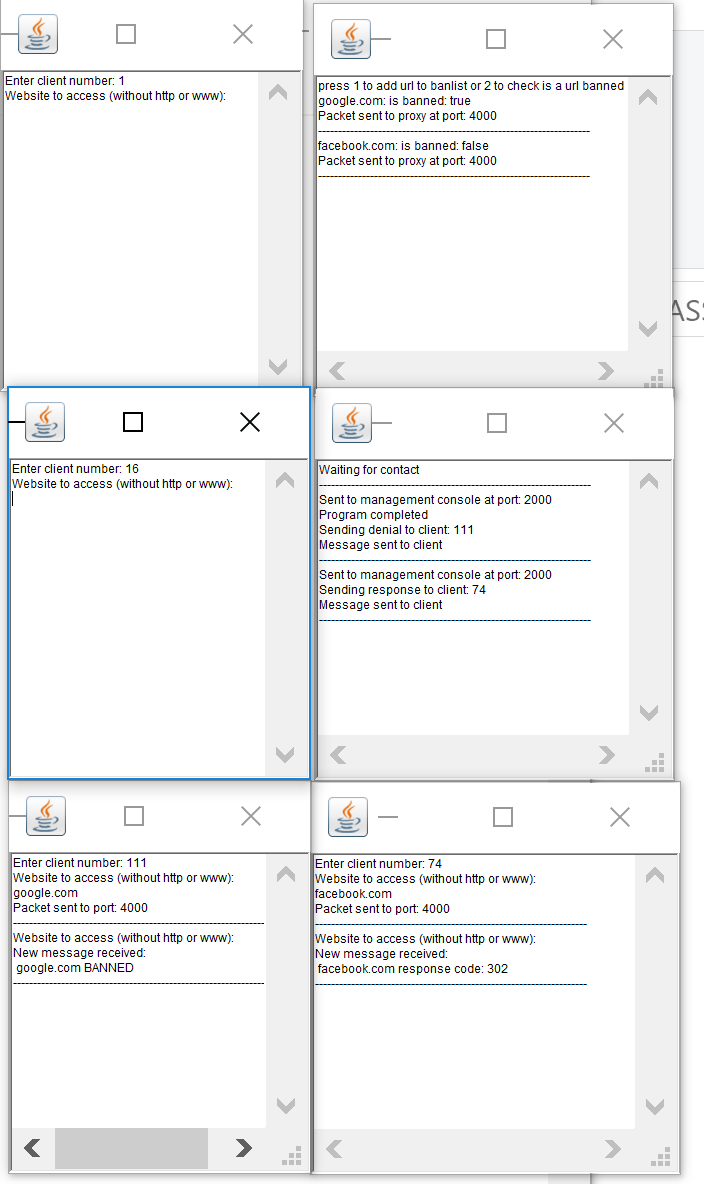
String content implements the packet content class. It returns the string and makes a datagram packet through.

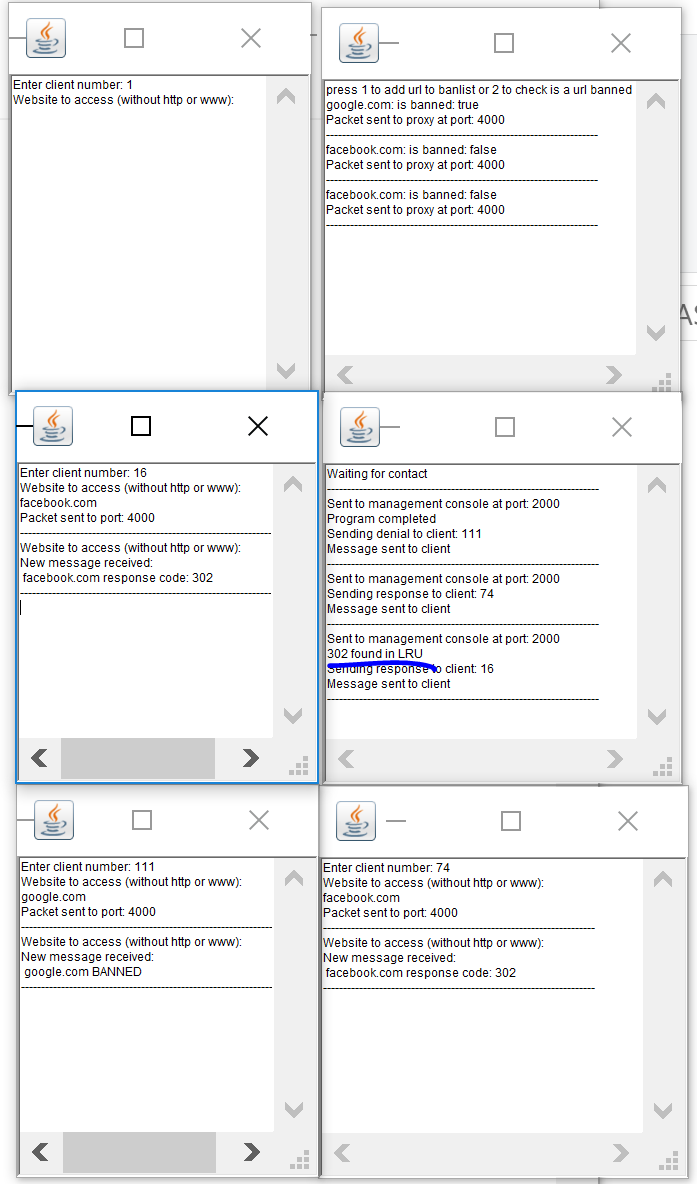
Node class:

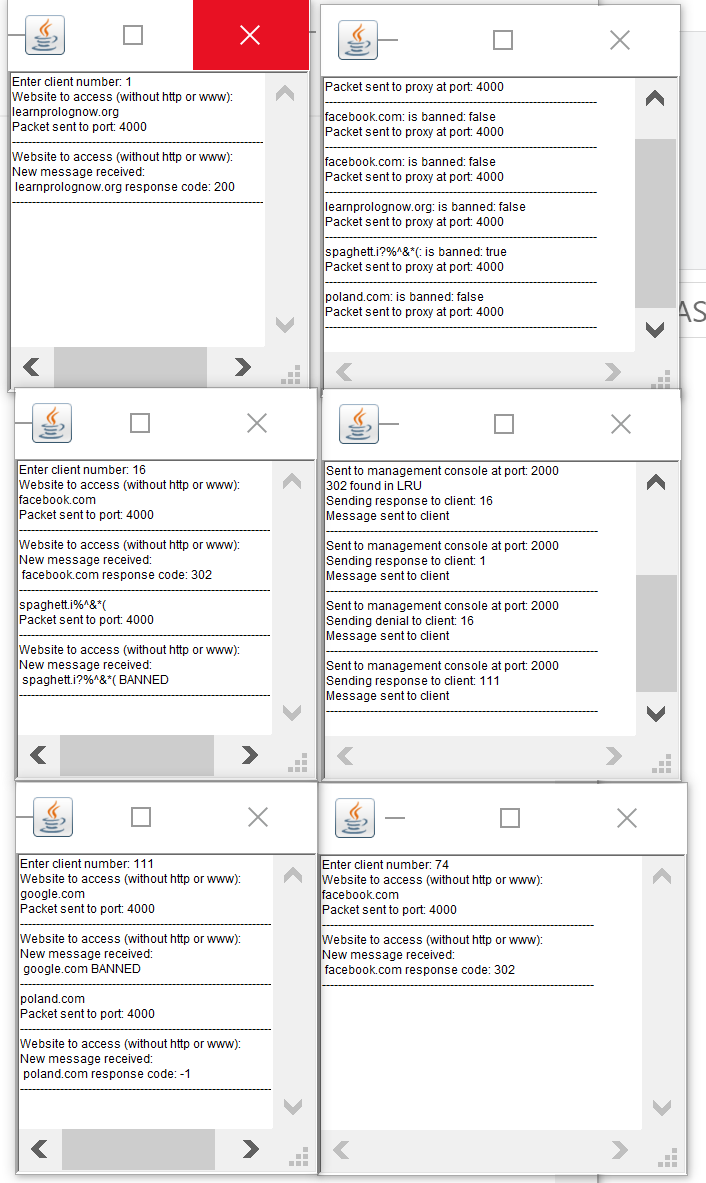
The listener function in the node class listens for incoming packets on a datagram socket and informs registered receivers about incoming packets. It listens for incoming packets and informs receivers upon arrival.

Sample snippets of program clients:





response found in LRU cache: (blue line)

**CODE dump:**

Please view my code on: <https://github.com/slow-J/CS3031-Telecoms> as it looks a bit ugly below

|  |  |
| --- | --- |
| Web\_Proxy\_Client | import java.net.DatagramSocket;  import java.io.IOException;  import java.net.DatagramPacket;  import java.net.InetSocketAddress;  import java.net.SocketTimeoutException;  import tcdIO.\*;  /\*\*  \* @author slowinsj  \*  \*/  public class Web\_Proxy\_Client extends Node  {  static final String DEFAULT\_DST\_NODE = "localhost";  static final int DEFAULT\_SRC\_PORT = 1000;  static final int DEFAULT\_DST\_PORT = 4000;  byte client\_no;  Terminal terminal;  InetSocketAddress dstAddress;  /\*\*  \* @param args  \*/  Web\_Proxy\_Client(Terminal terminal,byte client, int src\_port) throws SocketTimeoutException {  try  {  client\_no = client;  this.terminal = terminal;  socket = new DatagramSocket(src\_port);  listener.go();  } catch (java.lang.Exception e)  {  e.printStackTrace();  }  }  public void start() throws SocketTimeoutException  {  while (true)  {  DatagramPacket packet = null;  byte[] payload = null;  byte[] header = null;  byte[] buffer = null;  payload = (terminal.readString("Website to access (without http or www): \n")).getBytes();  header = new byte[PacketContent.HEADERLENGTH];  header[0] = (byte)client\_no;  header[1] = -1;  dstAddress = new InetSocketAddress(DEFAULT\_DST\_NODE, DEFAULT\_DST\_PORT);  buffer = new byte[header.length + payload.length];  System.arraycopy(header, 0, buffer, 0, header.length);  System.arraycopy(payload, 0, buffer, header.length, payload.length);  packet = new DatagramPacket(buffer, buffer.length, dstAddress);  // send packet to dest  try  {  socket.send(packet);  } catch (IOException e)  {  // TODO Auto-generated catch block  e.printStackTrace();  }  terminal.println("Packet sent to port: " + DEFAULT\_DST\_PORT);  terminal.println("--------------------------------------------------------------------");    }    }  public static void main(String[] args)  {  try  {  Terminal terminal = new Terminal("Client");  byte client\_no = terminal.readByte("Enter client number: ");  (new Web\_Proxy\_Client(terminal, client\_no,DEFAULT\_SRC\_PORT+client\_no)).start();  terminal.println("Program completed");  }  catch (java.lang.Exception e)  {  e.printStackTrace();  }  }    public synchronized void onReceipt(DatagramPacket packet)  {  StringContent content = new StringContent(packet);  terminal.println("New message received:\n " + content.toString());  terminal.println("--------------------------------------------------------------------");    this.notify();  }  } |
| Proxy\_Server | import java.io.IOException;  import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.HttpURLConnection;  import java.net.InetSocketAddress;  import java.net.MalformedURLException;  import java.net.ProtocolException;  import java.net.URL;  import tcdIO.\*;  /\*\*  \*  \*/  /\*\*  \* @author slowinsj  \*  \*/  public class Proxy\_Server extends Node  {  Terminal terminal;  InetSocketAddress dstAddress;  static final String DEFAULT\_DST\_NODE = "localhost";  static final int DEFAULT\_SRC\_PORT = 4000;  static final int DEFAULT\_CLIENT\_PORT = 1000;  static final int DEFAULT\_MANAGE\_PORT = 2000;  static LRUcache myLRU;  /\*\*  \* @param args  \*/    Proxy\_Server(Terminal terminal, int src\_port )  {  try  {  this.terminal = terminal;  socket = new DatagramSocket(src\_port);  listener.go();  } catch (java.lang.Exception e)  {  e.printStackTrace();  }  }  public static void main(String[] args)  {  // cache of capacity of 4  myLRU = new LRUcache(4);  try  {  Terminal terminal = new Terminal("Proxy\_Server");  (new Proxy\_Server(terminal,DEFAULT\_SRC\_PORT)).start();  terminal.println("Program completed");  } catch (java.lang.Exception e)  {  e.printStackTrace();  }  }  // when it receives packet  public synchronized void onReceipt(DatagramPacket packet)  {  byte[] buffer = packet.getData();  StringContent content = new StringContent(packet);  //terminal.println(content.toString());  byte client\_no = buffer[0];    // for testing terminal.println(client\_no+" client");    // notBan is -1 when not checked if banned  int notBan = buffer[1];  // for testing terminal.println(""+notBan);  if ((notBan)==-1) // came from client  {  dstAddress = new InetSocketAddress(DEFAULT\_DST\_NODE, DEFAULT\_MANAGE\_PORT);  terminal.println("Sent to management console at port: "+DEFAULT\_MANAGE\_PORT);  packet.setSocketAddress(dstAddress);    try  {  socket.send(packet);  } catch (IOException e)  {  e.printStackTrace();  }    }  else  {  DatagramPacket newPacket = null;  byte[] payload = null;  byte[] header = new byte[PacketContent.HEADERLENGTH];  buffer = null;  dstAddress = new InetSocketAddress(DEFAULT\_DST\_NODE, client\_no+DEFAULT\_CLIENT\_PORT);  if(notBan==1)//http request  {    String url = content.toString();  int responseCode;  //cache access  if(myLRU.checkIfInCache(url))  {  responseCode=myLRU.getResponse(url);  terminal.println(responseCode+" found in LRU");  }  else  {  String urlwithhttp = "http://"+content.toString();  responseCode = httprequest(urlwithhttp);  myLRU.addLRUnode(url, responseCode);  }    payload = (url+" response code: " + responseCode).getBytes();  buffer = new byte[header.length + payload.length];  System.arraycopy(header, 0, buffer, 0, header.length);  System.arraycopy(payload, 0, buffer, header.length, payload.length);  terminal.println("Sending response to client: " + client\_no);  newPacket = new DatagramPacket(buffer, buffer.length, dstAddress);  //send back to client    }  else // when something is banned  {    payload = (content.toString()+" BANNED").getBytes();  buffer = new byte[header.length + payload.length];  System.arraycopy(header, 0, buffer, 0, header.length);  System.arraycopy(payload, 0, buffer, header.length, payload.length);  terminal.println("Sending denial to client: " + client\_no);  newPacket = new DatagramPacket(buffer, buffer.length, dstAddress);  // send packet to client  }  try  {  //sending packet  socket.send(newPacket);  terminal.println("Message sent to client");  terminal.println("--------------------------------------------------------------------");    } catch (IOException e)  {  e.printStackTrace();  }  }        this.notify();  }  public synchronized void start() throws Exception  {  terminal.println("Waiting for contact");  terminal.println("--------------------------------------------------------------------");  this.wait();  }  public static int httprequest(String urlString)  {  System.out.println(urlString);  URL urlObject = null;  try  {  urlObject = new URL(urlString);  } catch (MalformedURLException e1)  {  // should never happen as this should be handled by management console  e1.printStackTrace();  }  HttpURLConnection connect = null;  try  {  connect = (HttpURLConnection) urlObject.openConnection();  } catch (IOException e1)  {  // TODO Auto-generated catch block  e1.printStackTrace();  }  try  {  connect.setRequestMethod("GET");  } catch (ProtocolException e)  {  // TODO Auto-generated catch block  e.printStackTrace();  }    int responseCode = -1;  try  {  responseCode = connect.getResponseCode();  } catch (IOException e)  {  e.printStackTrace();  }    System.out.println("response code is " + responseCode);  return responseCode;  }  } |
| Management\_Console | import java.net.DatagramSocket;  import java.io.BufferedWriter;  import java.io.File;  import java.io.FileWriter;  import java.io.IOException;  import java.net.DatagramPacket;  import java.net.InetSocketAddress;  import java.net.MalformedURLException;  import java.net.SocketTimeoutException;  import java.net.URISyntaxException;  import java.net.URL;  import java.nio.file.Files;  import java.nio.file.Path;  import java.nio.file.Paths;  import java.util.ArrayList;  import tcdIO.\*;  /\*\*  \* @author Jakub  \*  \*/  public class Management\_Console extends Node  {  static final int DEFAULT\_CLIENT\_PORT = 1000;  static final int DEFAULT\_SRC\_PORT = 2000;  static final int DEFAULT\_DST\_PORT = 4000;  static final String DEFAULT\_DST\_NODE = "localhost";    Terminal terminal;  InetSocketAddress dstAddress;  Management\_Console(Terminal terminal, int srcPort) throws SocketTimeoutException  {  try  {  // can only have one destination  dstAddress = new InetSocketAddress(DEFAULT\_DST\_NODE, DEFAULT\_DST\_PORT);  this.terminal = terminal;  socket = new DatagramSocket(srcPort);  listener.go();  } catch (java.lang.Exception e)  {  e.printStackTrace();  }  }  public synchronized void onReceipt(DatagramPacket packet)  {  byte[] buffer = packet.getData();  byte client\_no = buffer[0];  boolean banned = true;  StringContent content = new StringContent(packet);  String checkBan = content.toString();  //init ban list at start  //check now  if(checkIfValidURL("http://"+checkBan))  {  if(!checkIfBan("http://"+checkBan))  {  banned=false;  }  }    try  {  // add to banlist  DatagramPacket sendPacket = null;  byte[] payload = checkBan.getBytes();  byte[] header = new byte[PacketContent.HEADERLENGTH];  buffer = null;  // header[1] is where data is saved on ban or not ban  if (banned)  {  header[1] = 0;  } else  {  header[1] = 1;  }  header[0]= client\_no;  // send to source  buffer = new byte[header.length + payload.length];  System.arraycopy(header, 0, buffer, 0, header.length);  System.arraycopy(payload, 0, buffer, header.length, payload.length);  terminal.println(checkBan + ": is banned: " + banned);  sendPacket = new DatagramPacket(buffer, buffer.length, dstAddress);  // send packet to dest  socket.send(sendPacket);  terminal.println("Packet sent to proxy at port: " + DEFAULT\_DST\_PORT);  terminal.println("--------------------------------------------------------------------");  } catch (IOException e)  {  e.printStackTrace();  }  this.notify();  }  public void start() throws Exception  {  initBanList();  while (true)  {  int action = (terminal.readInt("press 1 to add url to banlist or 2 to check is a url banned\n"));  if (action==1)  {  String ban = "http://"+(terminal.readString("type website to ban without the www or http\n"));  if(checkIfValidURL(ban))  {  add2ban(ban);  }  else  {  terminal.println(ban+", not a valid url");  }  }  else if(action==2)  {  String ban = "http://"+(terminal.readString("type website to ban without the www or http\n"));  if(!checkIfValidURL(ban))  {  terminal.println(ban+", not a valid url");  }  else  {  if (checkIfBan(ban))  terminal.println(ban + ", is banned");  else  terminal.println(ban + ", is not banned");  }  }  else  {  terminal.println("Invalid selection");  }  terminal.println("--------------------------------------------------------------------");    }  }  /\*  \* @param url including http://  \*  \* checks to see if the syntax is valid or url is malformed  \*  \* @return true if valid url  \*/  private static boolean checkIfValidURL(String tryurl)  {  //checks if url valid  try  {  URL url1 = new URL(tryurl);  url1.toURI();  }  catch (MalformedURLException | URISyntaxException e)  {  //e.printStackTrace();  return false;  }  return true;  }  public static void initBanList()  {  //creates file if doesnt exist  try  {  File myFile;  myFile = new File("banlist.txt");  myFile.createNewFile();  } catch (IOException e)  {  e.printStackTrace();  }  }  public void add2ban(String banWord)  {  if (!checkIfBan(banWord))  {  // adds banWord to the banlist  BufferedWriter output = null;  try  {  output = new BufferedWriter(new FileWriter("banlist.txt", true));  } catch (IOException e2)  {  // TODO Auto-generated catch block  e2.printStackTrace();  }  try  {  //adds the banned url to the next line in the txt file  output.newLine();  output.append(banWord);  } catch (IOException e2)  {  // TODO Auto-generated catch block  e2.printStackTrace();  }  try  {  output.close();  } catch (IOException e2)  {  // TODO Auto-generated catch block  e2.printStackTrace();  }  terminal.println(banWord+", successfully added to banlist");  }  else  terminal.println(banWord+", already banned");  }  public static boolean checkIfBan(String cmp)  {  Path path = Paths.get("banlist.txt");  ArrayList<String> lines=null;  try  {  //each line of txt file to arraylist of strings  lines = (ArrayList<String>) Files.readAllLines(path);  } catch (IOException e)  {  e.printStackTrace();  return false;  }  for(int i=0;i<lines.size();i++)  {  if(lines.get(i).equals(cmp))  return true;  //true if arraylist contains the compared word  }  return false;  }  public static void main(String[] args)  {  try  {  Terminal terminal = new Terminal("Management\_Console");  (new Management\_Console(terminal,DEFAULT\_SRC\_PORT )).start();    terminal.println("Program completed");  } catch (java.lang.Exception e)  {  e.printStackTrace();  }  }  } |
| Node | import java.net.DatagramPacket;  import java.net.DatagramSocket;  import java.net.SocketException;  import java.util.concurrent.CountDownLatch;  public abstract class Node  {  static final int PACKETSIZE = 65536;  DatagramSocket socket;  Listener listener;  CountDownLatch latch;  Node()  {  latch = new CountDownLatch(1);  listener = new Listener();  listener.setDaemon(true);  listener.start();  }  public abstract void onReceipt(DatagramPacket packet);  /\*\*  \*  \* Listener thread  \*  \* Listens for incoming packets on a datagram socket and informs registered  \* receivers about incoming packets.  \*/  class Listener extends Thread  {  /\*  \* Telling the listener that the socket has been initialized  \*/  public void go()  {  latch.countDown();  }  /\*  \* Listen for incoming packets and inform receivers  \*/  public void run()  {  try {  latch.await();  // Endless loop: attempt to receive packet, notify receivers, etc  while (true) {  DatagramPacket packet = new DatagramPacket(new byte[PACKETSIZE], PACKETSIZE);  socket.receive(packet);  onReceipt(packet);  }  } catch (Exception e) {  if (!(e instanceof SocketException))  e.printStackTrace();  }  }  }  } |
| PacketContent | **import** java.net.DatagramPacket;  **public** **interface** PacketContent  {  **public** **static** **byte** ***HEADERLENGTH*** = 10;  **public** **String** **toString**();  **public** **DatagramPacket** **toDatagramPacket**();  } |
| StringContent | **import** java.net.DatagramPacket;  **public** **class** **StringContent** **implements** PacketContent  {  **String** string;  **public** **StringContent**(**DatagramPacket** packet)  {  **byte**[] **payload**;  **byte**[] **buffer**;  buffer = packet.getData();  payload = **new** **byte**[packet.getLength() - ***HEADERLENGTH***];  **System**.*arraycopy*(buffer, ***HEADERLENGTH***, payload, 0, packet.getLength() - ***HEADERLENGTH***);  string = **new** String(payload);  }  **public** **StringContent**(**String** string)  {  **this**.string = string;  }  **public** **String** **toString**()  {  **return** string;  }  **public** **DatagramPacket** **toDatagramPacket**()  {  **DatagramPacket** **packet** = **null**;  **byte**[] **buffer** = **null**;  **byte**[] **payload** = **null**;  **byte**[] **header** = **null**;  **try** {  payload = string.getBytes();  header = **new** **byte**[***HEADERLENGTH***];  buffer = **new** **byte**[header.length + payload.length];  **System**.*arraycopy*(payload, 0, buffer, ***HEADERLENGTH***, payload.length);  packet = **new** DatagramPacket(buffer, buffer.length);  } **catch** (**Exception** **e**) {  e.printStackTrace();  }  **return** packet;  }  } |
| LRUcache | **import** java.util.ArrayList;  **public** **class** **LRUcache**  {  **ArrayList**<LRUnode> list = **null**;  **int** capacity;    **public** **LRUcache**(**int** capacity)  {  **this**.list = **new** ArrayList<>();  **this**.capacity = capacity;  }  **public** **boolean** **checkIfInCache**(**String** cmpkey)  {  //doesnt make head as not accessed yet  **for**(**int** **i**=0;i<list.size();i++)  {  **if**(list.get(i).key.equals(cmpkey))  {  **return** **true**;  }  }  **return** **false**;  }  //for testing purposes only  **public** **void** **getList**()  {  **for**(**int** **i**=0; i<4;i++)  {  **System**.***out***.println(list.get(i).key+" "+list.get(i).value);  }  }  **public** **int** **getResponse**(**String** key)  {  **for**(**int** **i**=0;i<list.size();i++)  {  **if**(list.get(i).key.equals(key))  {  // store as temporary node before removing and adding as new head  **LRUnode** **tmp** = list.get(i);  list.remove(list.get(i));  //make head  list.add(0, tmp);  **return** tmp.value;  }  }  //shouldnt do this as checkInCache should be called before getResponse  **return** **Integer**.***MAX\_VALUE***;  }  **public** **void** **addLRUnode**(**String** cmpkey, **int** val)  {  **boolean** **trigger** = **false**;  **for**(**int** **i**=0;i<list.size();i++)  {  **if**(list.get(i).key.equals(cmpkey))  {  trigger=**true**;  **LRUnode** **tmp** = list.get(i);  list.remove(list.get(i));  //make head  list.add(0, tmp);  }  }  **if**(!trigger)  {  **if**(list.size()>=**this**.capacity)  {  list.remove(**this**.capacity-1);  }  **LRUnode** **tmp** = **new** LRUnode(cmpkey, val);  //make head  list.add(0, tmp);  }    }    } |
| LRUnode | **public** **class** **LRUnode**  {  **String** key;  **int** value;  **LRUnode** prev;  **LRUnode** next;  **public** **LRUnode**(**String** key, **int** value)  {  **this**.key = key;  **this**.value = value;  }  } |
| blacklist.txt | test  aasfagadg  Glad  ass.aa  http://gogle.com  http://google.aaaaaaaa  http://googl.com  http://ada.a.a.ads  http://aaa  http://aaa.jpg  http://google.wp.com  http://google.com  http://a  http://aa.pl  http://jakub.com  http://crab.com |