MANUAL TÉCNICO DEL PROXY UTN

El Proxy UTN fue desarrollado en Java con el JDK1.1.8, se emplearon los paquetes:

- java.net .- Para el manejo de comunicaciones de red.
- java.io.- Para manejo de entrada/salida.
- java.util.- Para utilizar funciones de utilidad que complementar a otros paquetes.
- java.awt.- Para manejar la interfaz de usuario.

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.awt.*;
```

Este comprende básicamente tres clases que son:

- ServerP
- HttpServer
- HttpConnection

ServerP.- Esta es la clase es desde la que se inicia el servidor, es una clase que hereda las funcionalidades de la clase Frame, con el fin de poder utilizar un contenedor que permita posteriormente incluir componentes dentro del mismo.

Esta clase contiene tres métodos:

- El método constructor
- El método handleEvent
- El método main

Método constructor.- Este método crea la ventana al llamar al constructor de la clase padre, luego añade un Label en el Frame y lo muestra, con esto se inicia el programa.

```
public class ServerP extends Frame{

  Label campo = new Label("Servidor iniciado ....");
 public Server13()
 {
   super("Servidor Proxy UTN");
   add(campo);
   resize(200,80);
   show();
 }
```

Método handleEvent.- Este método devuelve un boolean como resultado de su operación, este método maneja las acciones que ocurren en la ventana, en este caso se tiene la opción de cerrar la ventana desde la esquina superior derecha.

Al hacer click en esta esquina, se liberan los recursos que están siendo utilizados en ese momento y se procede a detener el servidor; sino se ejecuta ninguna acción, el servidor seguirá ejecutándose.

```
public boolean handleEvent(Event event)
 {
   if(event.id == Event.WINDOW_DESTROY)
   {
     hide();
     dispose();
     System.exit(0);
     return true;
   }
     return super.handleEvent(event);
 }
```

Método main.- Este es el método principal, pues este es el que inicia el servidor y los objetos que este necesita para su funcionamiento.

Se crea un objeto de la clase ServerP, para que se muestre el Frame, luego se crea un objeto de la clase HttpServer

```
public static void main(String[] args){
      ServerP nuevo = new ServerP();
     HttpServer httpServerThread = new HttpServer();
  }
}
```

HttpServer.- Esta clase es una extensión de Thread y a través del método start llama al método run e inicializa un ServerSocket en el puerto 8080, el cual viene a constituir el puerto del servidor que está esperando una conexión por parte de un cliente.

El momento que un cliente se conecta con el servidor se crea un nuevo objeto de la clase HttpConnection.

```
class HttpServer extends Thread{

  HttpServer(){
    start();
  }
  /

  public void run(){
    try{
        ServerSocket serverSocket = new ServerSocket(8080);
        System.out.println("Servidor escuchando en el puerto 8080");

    while(true)
      new HttpConnection(serverSocket.accept());
    }catch(IOException e){System.out.println(e);}
  }
}
```

HttpConnection.- Esta clase al igual que HttpServer es una extensión de Thread. Aquí se realiza la declaración de variables globales que se emplearán en el proceso de conexión con el cliente y con los servidores reales de Internet, que son con los cuales los usuarios internos desean comunicarse.

La clase HttpConnection maneja cinco métodos que son:

- Método constructor
- Método Logs
- Método AnalizaIP

- Método Analiza URL
- Método run

```
class HttpConnection extends Thread{
   Socket socket;
   BufferedReader inputStream = null;
   PrintWriter outputStream = null;
   DataOutputStream byteOutput = null;
   URL url = null;
   InputStream in = null;
   InetAddress address = null;
   String request = null;
```

Método constructor.- El constructor de HttpConnection ata un socket, cambia la prioridad de funcionamiento e inicia el método run heredado de la clase Thread.

```
HttpConnection(Socket socket){
   this.socket = socket;
   setPriority( NORM_PRIORITY-1 );
   start();
 }
 /
```

Método Logs.- Este método genera los archivos de logs de la aplicación, no devuelve ningún valor es un método de tipo void, espera como parámetro un valor entero, el cual indica a que archivo de log tiene que añadir información.

En primer lugar se obtiene la dirección IP del equipo que solicitó la conexión, luego se abren los flujos de salida sobre el archivo apropiado y se procede a añadir la información correspondiente.

```
public void Logs(int log)
 {
    Date fecha = new Date();
    String valorip;
    address = socket.getInetAddress();
    valorip = address.getHostAddress();
    FileWriter fw = null;
    BufferedWriter bw = null;
    PrintWriter cadena = null;
    switch (log)
    {
      case 1: try
```

```
                {
                  fw = new FileWriter("d:/proxy_utn/dominio_deny_log.txt",true);
                  bw = new BufferedWriter(fw);
                  cadena = new PrintWriter(bw);
                  cadena.println(fecha.toString() + "  " + valorip + "  " + request);
                  cadena.close();
                  bw.close();
            fw.close();
                }catch (java.io.FileNotFoundException fnfx){
                  System.out.println("No existe el archivo dominio_deny.txt");}
                  catch (java.io.IOException ioex) {}
                 break;
    case 2: try
                {
                  fw = new FileWriter("d:/proxy_utn/ip_deny_log.txt",true);
                  bw = new BufferedWriter(fw);
                  cadena = new PrintWriter(bw);
                  cadena.println(fecha.toString() + "  " + valorip + "  " + request);
                  cadena.close();
            bw.close();
            fw.close();
                }catch (java.io.FileNotFoundException fnfx){
                  System.out.println("No existe el archivo ip_deny.txt");}
                catch (java.io.IOException ioex) {}
                break;
    case 3: try
                {
                  fw = new FileWriter("d:/proxy_utn/access_log.txt",true);
                  bw = new BufferedWriter(fw);
                  cadena = new PrintWriter(bw);
                  cadena.println(fecha.toString() + "  " + valorip + "  " + request);
                  cadena.close();
            bw.close();
           fw.close();
                }catch (java.io.FileNotFoundException fnfx){
                  System.out.println("No existe el archivo access_log.txt");}
                catch (java.io.IOException ioex) {}
                break;
  }
}
```

Método AnalizaIP.- Este método analiza la dirección IP del equipo que solicita la conexión, luego busca en el archivo ip_deny y si lo encuentra devuelve uno indicando que la dirección no tiene privilegios de conexión. Si no encuentra la dirección en el archivo ip_deny, devuelve dos.

```
public int AnalizaIP(){
  String valorip;
  String texto = new String("");
  String s = new String("");
  try
  {
    FileReader fr = new FileReader("d:/proxy_utn/ip_deny.txt");
    BufferedReader entrada = new BufferedReader(fr);
    address = socket.getInetAddress();
    valorip = address.getHostAddress();
    while(entrada.ready())
```

```
    {
       texto = entrada.readLine();
       s = texto.trim();
       if(s.equals(valorip))
         return(1);
    }
    entrada.close();
    fr.close();
  }catch(java.io.FileNotFoundException fnfex) {
    System.out.println("No existe el archivo ip_deny.txt");
  }
   catch(java.io.IOException ioex) {}
   return(2);  //devuelve 2 si no encuentra la direccion en el archivo
}
```

Método AnalizaIP.- Este método analiza la URL solicitada por cliente, recibe como parámetro el dominio solicitado. Realiza una búsqueda en el archivo dominio_deny, si encuentra el dominio devuelve uno, caso contrario devuelve dos.

```
public int AnalizaURL(String dominio){
  String texto = new String("");
  String s = new String("");

  StringTokenizer stringTokenizer = new StringTokenizer(dominio);
  if((stringTokenizer.countTokens() >= 2) && stringTokenizer.nextToken().equals("GET"))
  {
     stringTokenizer.nextToken("//");
     dominio = stringTokenizer.nextToken();
  }

  try
  {
    FileReader fr = new FileReader("d:/proxy_utn/dominio_deny.txt");
    BufferedReader entrada = new BufferedReader(fr);
    while(entrada.ready())
    {
       texto = entrada.readLine();
       s = texto.trim();
       if(s.equals(dominio))
         return(1);
    }
    entrada.close();
    fr.close();
  }catch(java.io.FileNotFoundException fnfex) {
    System.out.println("No existe el archivo domionio_deny.txt");
  }
   catch(java.io.IOException ioex) {}
   return(2);
}
```

Método run.- Este método genera los flujos de entrada y salida sobre el socket y sobre el url solicitado.

Se genera flujos de bytes sobre el socket, con el fin de poder leer las solicitudes del cliente y escribir las respuestas que los servidores de internet devuelven al proxy como resultado de las peticiones de los clientes de la red interna.

Se crea un flujo sobre el objeto url con el objetivo de poder leer la información que el servidor nos devuelve como resultado de la petición realizada.

En primer lugar se procede a realizar el análisis de la dirección IP del equipo solicitante con el fin de determinar si puede o no conectarse a internet. En caso de ser negativa la respuesta, es decir, que la dirección del equipo se encuentra en el archivo ip_deny, se devuelve al cliente una respuesta desde el servidor proxy, indicándole que no puede conectarse a Internet.

Luego se procede a analizar la dirección URL solicitada por el cliente, igual que en el caso de direcciones, si la URL está negada, se procede a enviar al cliente una notificación de que el acceso está restringido.

Finalmente si el acceso está permitido se crea un objeto URL que contiene como parámetro la petición del cliente, se abre un openStream sobre el objeto URL y de ahí se procede a leer el contenido del InputStream en el cual se almacena el contenido del URL solicitado.

Una vez que se termina de leer el InputStream, se deben cerrar todos los flujos que se abrieron anteriormente con el fin de no consumir recursos que pueden ser necesarios para otras conexiones.

Por las conexiones solicitadas se generan los archivos de log correspondientes, en los cuales se almacenan las fechas, direcciones IP de los clientes y los dominios solicitados.

```
public void run(){
int resp=0;
try{

inputStream = new DataInputStream(socket.getInputStream());

outputStream = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()),true);

byteOutput = new DataOutputStream(socket.getOutputStream());

request = inputStream.readLine();

while(request.length()>0)
{
  header.addElement(request);
  request = inputStream.readLine();
}

resp = AnalizaIP();
if(resp==1)
{
 outputStream.println("<html><body><body bgcolor=Teal
Text=White><H3><title>Proxy UTN </title></h3><marquee width=
150><Center><I>ADVERTENCIA</I></center></marquee><br><hr></hr><strong
color= white ><I>IP restringida</I></h2></CENTER><br><strong color=
white ><I>Por favor No intente acceder
nuevamente</I></h2></CENTER><br></body></html>");
 System.out.println("Socket cerrado por IP bloqueado");
 Logs(2);
 socket.close();
 outputStream.close();
 resp=-1;
}
else
{
resp=0;
StringTokenizer stringTokenizer = new StringTokenizer((String)
header.elementAt(0));
metodo = stringTokenizer.nextToken();
request = stringTokenizer.nextToken();
try
{
  url=new URL(request);
} catch (MalformedURLException mue){
  System.out.println(mue.getMessage());}

resp = AnalizaURL();
if(resp==1)
```

```
{
outputStream.println("<HTML><BODY><P>P…gina prohibida, por   favor no
intente acceder<P></BODY></HTML>");
System.out.println("Socket cerrado por p gina bloqueada");
Logs(1);
socket.close();
outputStream.close();
resp=-1;
}

else
{
resp=0;
resp = AnalizaProtocolo();

if(resp==1)
{
outputStream.println("<HTML><BODY><P>Protocolo no
permitido<P></BODY></HTML>");
System.out.println("Socket cerrado por protocolo no permitido");
Logs(1);
socket.close();
outputStream.close();
resp=-1;
}
else
{
  host = url.getHost();
  port = url.getPort();
  if(port == -1)
     port = 80;
  System.out.println("Host "+host);
  System.out.println("Puerto "+port);
  socket1 = new Socket(host,port);
  System.out.println("Socket creado");

  out = socket1.getOutputStream();
  System.out.println("Flujo de salida creado");

  urlconect=url.openConnection();

  int a = header.size();
  System.out.println("Valor de a "+a);
  String tmp;
  for(int h=0; h<a; h++)
  {
     tmp = (String)header.elementAt(h);
     out.write((tmp+"\r\n").getBytes());
     System.out.println("Valor de "+ h +" "+ tmp);
     out.flush();
   }
   out.write("\r\n".getBytes());
   out.flush();

   int size = getHeaderFieldInt("content-length",-1);
   System.out.println("Size " +size);
   if(size != -1)
   {
```

```
        byte post_data[];
        post_data=new byte[size];
        System.out.println("Creado post_data");
        int hi = inputStream.read(post_data);
        System.out.println("Valor de hi "+hi);
        String tmp2 = post_data.toString();
        System.out.println("Valor de tmp2 "+tmp2);
        System.out.println("Iniciando envio");
        out.write(post_data);
        out.flush();
        out.write("\r\n".getBytes());
        out.flush();
        System.out.println("Terminado envio");
        out.flush();
       }

      in = socket1.getInputStream();

     request1 = urlconect.getHeaderField(0);
     System.out.println(request1);

      int i=0;
      i = in.read();
      while(i != -1)
      {
        byteOutput.write(i);
        i = in.read();
      }
      outputStream.close();
      byteOutput.flush();
      in.close();
      out.close();
      Logs(3);
      System.out.println("Cerrando " +request);
      System.out.println("Buffer cerrado fin de main");
   }   }
}
}catch(IOException e)
 {
   System.out.println( "I/O error " + e );
   excepciones = e.toString();
   System.out.println(excepciones);
   if(excepciones.equals("java.net.ConnectException: Connection
   refused"))
   outputStream.println("<HTML><BODY><P>Conexion rechazada, servidor no
   responde<P></BODY></HTML>");
   else
         outputStream.println("<HTML><BODY><P>Servidor
         desconocido<P></BODY></HTML>");
   try
   {
   socket.close();
   outputStream.close();
   byteOutput.flush();
   System.out.println("Socket cerrado por servidor bajo");
   }catch(IOException evt){System.out.println(evt);}
 }
}}
```