**Skeleton Java Code for the Proxy Server**

**ProxyCache.java**

|  |
| --- |
| /\*\*  \* ProxyCache.java - Simple caching proxy  \*  \* $Id: ProxyCache.java,v 1.3 2004/02/16 15:22:00 kangasha Exp $  \*  \*/  import java.net.\*;  import java.io.\*;  import java.util.\*;  public class ProxyCache {  /\*\* Port for the proxy \*/  private static int port;  /\*\* Socket for client connections \*/  private static ServerSocket socket;  /\*\* Create the ProxyCache object and the socket \*/  public static void init(int p) {  port = p;  try {  socket = new ServerSocket(port);  } catch (IOException e) {  System.out.println("Error creating socket: " + e);  System.exit(-1);  }  }  public static void handle(Socket client) {  Socket server = null;  HttpRequest request = null;  HttpResponse response = null;  /\* Process request. If there are any exceptions, then simply  \* return and end this request. This unfortunately means the  \* client will hang for a while, until it timeouts. \*/  /\* Read request \*/  try {  BufferedReader fromClient = **new** BufferedReader(**new** InputStreamReader(client.getInputStream()));  request = **new** HttpRequest(fromClient);  } catch (IOException e) {  System.out.println("Error reading request from client: " + e);  return;  }  /\* Send request to server \*/  try {  /\* Open socket and write request to socket \*/  server = **new** Socket(request.getHost(), request.getPort());;  DataOutputStream toServer = **new** DataOutputStream(server.getOutputStream());;  toServer.writeBytes(request.toString());  } catch (UnknownHostException e) {  System.out.println("Unknown host: " + request.getHost());  System.out.println(e);  return;  } catch (IOException e) {  System.out.println("Error writing request to server: " + e);  return;  }  /\* Read response and forward it to client \*/  try {  DataInputStream fromServer = **new** DataInputStream(server.getInputStream());  response = **new** HttpResponse(fromServer);  DataOutputStream toClient = **new** DataOutputStream(client.getOutputStream());  toClient.writeBytes(response.toString());  toClient.write(response.body);  /\* Write response to client. First headers, then body \*/  client.close();  server.close();  /\* Insert object into the cache \*/  /\* Fill in (optional exercise only) \*/  } catch (IOException e) {  System.out.println("Error writing response to client: " + e);  }  }  /\*\* Read command line arguments and start proxy \*/  public static void main(String args[]) {  int myPort = 0;    try {  myPort = Integer.parseInt(args[0]);  } catch (ArrayIndexOutOfBoundsException e) {  System.out.println("Need port number as argument");  System.exit(-1);  } catch (NumberFormatException e) {  System.out.println("Please give port number as integer.");  System.exit(-1);  }    init(myPort);  /\*\* Main loop. Listen for incoming connections and spawn a new  \* thread for handling them \*/  Socket client = null;    while (true) {  try {  client = *socket*.accept();  handle(client);  } catch (IOException e) {  System.out.println("Error reading request from client: " + e);  /\* Definitely cannot continue processing this request,  \* so skip to next iteration of while loop. \*/  continue;  }  }  }  } |

**HttpRequest.java**

|  |
| --- |
| /\*\*  \* HttpRequest - HTTP request container and parser  \*  \* $Id: HttpRequest.java,v 1.2 2003/11/26 18:11:53 kangasha Exp $  \*  \*/  import java.io.\*;  import java.net.\*;  import java.util.\*;  public class HttpRequest {  /\*\* Help variables \*/  final static String CRLF = "\r\n";  final static int HTTP\_PORT = 80;  /\*\* Store the request parameters \*/  String method;  String URI;  String version;  String headers = "";  /\*\* Server and port \*/  private String host;  private int port;  /\*\* Create HttpRequest by reading it from the client socket \*/  public HttpRequest(BufferedReader from) {  String firstLine = "";  try {  firstLine = from.readLine();  } catch (IOException e) {  System.out.println("Error reading request line: " + e);  }  String[] tmp = firstLine.split(" ");  method = tmp[0];  URI = tmp[1];  version = tmp[2];  System.out.println("URI is: " + URI);  if (!method.equals("GET")) {  System.out.println("Error: Method not GET");  }  try {  String line = from.readLine();  while (line.length() != 0) {  headers += line + CRLF;  /\* We need to find host header to know which server to  \* contact in case the request URI is not complete. \*/  if (line.startsWith("Host:")) {  tmp = line.split(" ");  if (tmp[1].indexOf(':') > 0) {  String[] tmp2 = tmp[1].split(":");  host = tmp2[0];  port = Integer.parseInt(tmp2[1]);  } else {  host = tmp[1];  port = HTTP\_PORT;  }  }  line = from.readLine();  }  } catch (IOException e) {  System.out.println("Error reading from socket: " + e);  return;  }  System.out.println("Host to contact is: " + host + " at port " + port);  }  /\*\* Return host for which this request is intended \*/  public String getHost() {  return host;  }  /\*\* Return port for server \*/  public int getPort() {  return port;  }  /\*\*  \* Convert request into a string for easy re-sending.  \*/  public String toString() {  String req = "";  req = method + " " + URI + " " + version + CRLF;  req += headers;  /\* This proxy does not support persistent connections \*/  req += "Connection: close" + CRLF;  req += CRLF;    return req;  }  } |

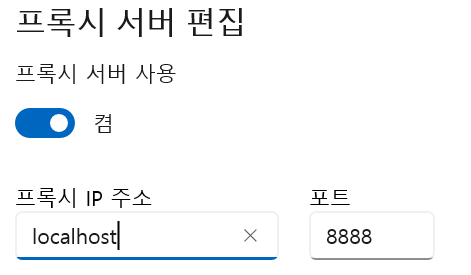
**HttpResponse.java**

|  |
| --- |
| /\*\*  \* HttpResponse - Handle HTTP replies  \*  \* $Id: HttpResponse.java,v 1.2 2003/11/26 18:12:42 kangasha Exp $  \*  \*/  import java.io.\*;  import java.net.\*;  import java.util.\*;  public class HttpResponse {  final static String CRLF = "\r\n";  /\*\* How big is the buffer used for reading the object \*/  final static int BUF\_SIZE = 8192;  /\*\* Maximum size of objects that this proxy can handle. For the  \* moment set to 100 KB. You can adjust this as needed. \*/  final static int MAX\_OBJECT\_SIZE = 100000;  /\*\* Reply status and headers \*/  String version;  int status;  String statusLine = "";  String headers = "";  /\* Body of reply \*/  byte[] body = new byte[MAX\_OBJECT\_SIZE];  /\*\* Read response from server. \*/  public HttpResponse(DataInputStream fromServer) {  /\* Length of the object \*/  int length = -1;  boolean gotStatusLine = false;  /\* First read status line and response headers \*/  try {  String line = fromServer.readLine();  while (line.length() != 0) {  if (!gotStatusLine) {  statusLine = line;  gotStatusLine = true;  } else {  headers += line + CRLF;  }  /\* Get length of content as indicated by  \* Content-Length header. Unfortunately this is not  \* present in every response. Some servers return the  \* header "Content-Length", others return  \* "Content-length". You need to check for both  \* here. \*/  if (line.startsWith("Content-Length") ||  line.startsWith("Content-length")) {  String[] tmp = line.split(" ");  length = Integer.parseInt(tmp[1]);  }  line = fromServer.readLine();  }  } catch (IOException e) {  System.out.println("Error reading headers from server: " + e);  return;  }  try {  int bytesRead = 0;  byte buf[] = new byte[BUF\_SIZE];  boolean loop = false;  /\* If we didn't get Content-Length header, just loop until  \* the connection is closed. \*/  if (length == -1) {  loop = true;  }    /\* Read the body in chunks of BUF\_SIZE and copy the chunk  \* into body. Usually replies come back in smaller chunks  \* than BUF\_SIZE. The while-loop ends when either we have  \* read Content-Length bytes or when the connection is  \* closed (when there is no Connection-Length in the  \* response. \*/  while (bytesRead < length || loop) {  /\* Read it in as binary data \*/  int res = fromServer.read(buf, 0, ***BUF\_SIZE***);  if (res == -1) {  break;  }  /\* Copy the bytes into body. Make sure we don't exceed  \* the maximum object size. \*/  for (int i = 0;  i < res && (i + bytesRead) < MAX\_OBJECT\_SIZE;  i++) {  /\* Fill in \*/  }  bytesRead += res;  }  } catch (IOException e) {  System.out.println("Error reading response body: " + e);  return;  }  }  /\*\*  \* Convert response into a string for easy re-sending. Only  \* converts the response headers, body is not converted to a  \* string.  \*/  public String toString() {  String res = "";  res = statusLine + CRLF;  res += headers;  res += CRLF;    return res;  }  } |

과정  
1. “fill in” 부분 채워넣기

2. 프록시 설정

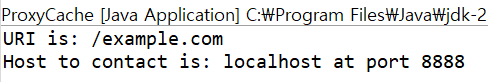
IP: localhost, port: 8888



3. ProxyServer를 실행시킨 상태로 웹 브라우저에 localhost:8888/example.com 입력

4. console에 출력된 값 확인

결과



위의 사진에서 보이는 곳 까지만 console 에 출력되고 나머진 출력되지 않았다.

브라우저는 무한 로딩(대기)에 걸렸으며 proxyserver도 종료되지 않았다.

오랜 시간 동안 구글링을 통해 디버깅 (URI is: /example.com Host to contact is: localhost at port 8888 Request read: GET /example.com HTTP/1.1 Host: localhost:8888) 도 해보고 flush() 라는 함수도 알게 되어 사용해보았지만 결과가 달라지지 않았다. 결론적으로 실패하였지만 **Optional Exercises**라도 진행해보았다.

**Optional Exercises**

1. **위의 코드는 status code: 404 와 같은 오류 상태를 처리하지 않는다. 만약 오류 상태라면 client가 요청한 객체가 없을 때, 코드 상 문제가 발생할 수 있다. 따라서 이를 방지하기 위해서는 status code로 오류 상태를 감지하도록 하는 코드를 추가하면 된다.**

**String statusLine = "";**

**String[] statusParts = statusLine.split(" ");**

**if (status == 404) { body = new byte[0]; return; }**

**해당 코드를 알맞은 위치에 추가하면 된다.**

1. **위의 코드는 GET만 지원하기 때문에 POST 방식도 지원하기 위해서는 별도의 코드가 필요하다.**

**String body = "";**

**if ("POST".equals(method) {   
int bytesRead = 0;   
while (bytesRead < body.length) {  
 int result = rawInputStream.read(body, bytesRead, body.length - bytesRead);   
if (result == -1) break; bytesRead += result;  
 }**