Sockets for Servers

Hyang-Won Lee
Department of Computer Science and Engineering
Konkuk University
leehw@konkuk.ac.kr
https://sites.google.com/view/leehwko/

ServerSocket Class

- Socket class needs host and port to open connection

```
public Socket(String host, int port) throws UnknownHostException, IOException
public Socket(InetAddress host, int port) throws IOException
```

- Server doesn't know who will request connection
  - server should keep listening to a bound port
  - Socket class cannot be used
- ServerSocket class contains
  - everything needed to write servers in Java
  - methods that listen for connections, methods that configure various server socket options,…

Constructing Server Sockets

- Constructors

```
public ServerSocket(int port) throws BindException, IOException
public ServerSocket(int port, int queueLength)
    throws BindException, IOException
public ServerSocket(int port, int queueLength, InetAddress bindAddress)
    throws IOException
public ServerSocket() throws IOException
```

length of queue used to hold incoming connection requests

- Constructing without binding

```
public void bind(SocketAddress endpoint) throws IOException
public void bind(SocketAddress endpoint, int queueLength) throws IOException
```

```
ServerSocket ss = new ServerSocket();
// set socket options...
SocketAddress  http = new InetSocketAddress(80);
ss.bind(http);
```

Example

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

public class LocalPortScanner {

  public static void main(String[] args) {

    for (int port = 1; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on the port
        ServerSocket server = new ServerSocket(port);
      } catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
      }
    }
  }
}
```

Getting Info

- Methods

```
public InetAddress getInetAddress()
public int getLocalPort()
```

- Example

```
import java.io.*;
import java.net.*;
public class RandomPort {

  public static void main(String[] args) {
    try {
      ServerSocket server = new ServerSocket(0);
      System.out.println("This server runs on port "
          + server.getLocalPort());
    } catch (IOException ex) {
      System.err.println(ex);
    }
  }
}
```

let OS choose port

Basic Life Cycle of Server Program

- step1 ◉ A new ServerSocket is created on a particular port using a ServerSocket() constructor
- step2 ◉ The ServerSocket listens for incoming connection attempts on that port using its accept() method
  - accept() blocks until a client attempts to make a connection, at which point accept() returns a Socket object connecting client and server
- step3 ◉ Either Socket's getInputStream(), getOutputStream() method, or both are called to get input and output streams
- step4 ◉ Server and client interact according to agreed-upon protocol
- step5 ◉ Server, client, or both close connection
- step6 ◉ Server returns to step 2 and waits for next connection

Example

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

public class DaytimeServer {

  public final static int PORT = 13;

  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try (Socket connection = server.accept()) {
         Writer out = new OutputStreamWriter(connection.getOutputStream());
         Date now = new Date();
         out.write(now.toString() +"\r\n");
         out.flush();
         connection.close();
       } catch (IOException ex) {}
     }
   } catch (IOException ex) {
     System.err.println(ex);
   }
  }
}
```

- Exercise
  - run the server and use DaytimeClient for testing the server
  - write multi-threaded client to overload the server

Multithreaded Servers

- Problem with DaytimeServer
  - when there is a slow client, what happens?

- What are the options?
  - create a new process to handle each connection? (Old-fashioned Unix)
    - slow down too quickly for a few hundreds of connections
  - spawn a new thread for each connection😀

Example

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

public class MultithreadedDaytimeServer {

  public final static int PORT = 13;

  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
         Socket connection = server.accept();
         Thread task = new DaytimeThread(connection);
         task.start();
       } catch (IOException ex) {}
     }
    } catch (IOException ex) {
      System.err.println("Couldn't start server");
    }
  }

  private static class DaytimeThread extends Thread {

    private Socket connection;

    DaytimeThread(Socket connection) {
      this.connection = connection;
    }

    @Override
    public void run() {
      try {
        Writer out = new OutputStreamWriter(connection.getOutputStream());
        Date now = new Date();
        out.write(now.toString() +"\r\n");
        out.flush();
      } catch (IOException ex) {
        System.err.println(ex);
      } finally {
        try {
          connection.close();
        } catch (IOException e) {
          // ignore;
        }
      }
    }
  }
}
```

- Exercise
  - compare with single-threaded server

Discussion

- What are the limitations of the code in the previous slide?
  - Too many threads can be spawned for servers with high volume of requests
  - Vulnerable to DoS attack

Example: Thread Pool

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

public class PooledDaytimeServer {
public final static int PORT = 13;

public static void main(String[] args) {

 ExecutorService pool = Executors.newFixedThreadPool(50);

 try (ServerSocket server = new ServerSocket(PORT)) {
   while (true) {
     try {
       Socket connection = server.accept();
       Callable<Void> task = new DaytimeTask(connection);
       pool.submit(task);
     } catch (IOException ex) {}
   }
  } catch (IOException ex) {
    System.err.println("Couldn't start server");
  }
}
```

```java
private static class DaytimeTask implements Callable<Void> {

  private Socket connection;

  DaytimeTask(Socket connection) {
    this.connection = connection;
  }

  @Override
  public Void call() {
    try {
      Writer out = new OutputStreamWriter(connection.getOutputStream());
      Date now = new Date();
      out.write(now.toString() +"\r\n");
      out.flush();
    } catch (IOException ex) {
      System.err.println(ex);
    } finally {
      try {
        connection.close();
      } catch (IOException e) {
        // ignore;
      }
    }
    return null;
  }
 }
}
```

- Exercise
  - compare with MultithreadedDaytimeServer

Writing to Servers with Sockets

- Server needs InputStream to read data from (client)
- Example skipped (too early to discuss Selector)

```
import java.nio.*;
import java.nio.channels.*;
import java.net.*;
import java.util.*;
import java.io.IOException;
public class EchoServer {

  public static int DEFAULT_PORT = 7;

  public static void main(String[] args) {

    int port;
    try {
      port = Integer.parseInt(args[0]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }
    System.out.println("Listening for connections on port " + port);
```

```
ServerSocketChannel serverChannel;
Selector selector;
try {
  serverChannel = ServerSocketChannel.open();
  ServerSocket ss = serverChannel.socket();
  InetSocketAddress address = new InetSocketAddress(port);
  ss.bind(address);
  serverChannel.configureBlocking(false);
  selector = Selector.open();
  serverChannel.register(selector, SelectionKey.OP_ACCEPT);
} catch (IOException ex) {
  ex.printStackTrace();
  return;
}
```

Example (contd.)

skipped (too early to discuss Selector)

```
while (true) {
  try {
    selector.select();
  } catch (IOException ex) {
    ex.printStackTrace();
    break;
  }

  Set<SelectionKey> readyKeys = selector.selectedKeys();
  Iterator<SelectionKey> iterator = readyKeys.iterator();
  while (iterator.hasNext()) {
    SelectionKey key = iterator.next();
    iterator.remove();
    try {
      if (key.isAcceptable()) {
        ServerSocketChannel server = (ServerSocketChannel) key.channel();
        SocketChannel client = server.accept();
        System.out.println("Accepted connection from " + client);
        client.configureBlocking(false);
        SelectionKey clientKey = client.register(
            selector, SelectionKey.OP_WRITE | SelectionKey.OP_READ);
        ByteBuffer buffer = ByteBuffer.allocate(100);
        clientKey.attach(buffer);
      }
      if (key.isReadable()) {
        SocketChannel client = (SocketChannel) key.channel();
        ByteBuffer output = (ByteBuffer) key.attachment();
        client.read(output);
      }
      if (key.isWritable()) {
        SocketChannel client = (SocketChannel) key.channel();
        ByteBuffer output = (ByteBuffer) key.attachment();
        output.flip();
        client.write(output);
        output.compact();
      }
    } catch (IOException ex) {
      key.cancel();
      try {
        key.channel().close();
      } catch (IOException cex) {}
    }
  }
}
```

- Exercise
  - Write an EchoClient

Exercise

- Write an EchoClient
  - send a string (from keyboard input) to server waiting on port 7
  - print the received (from server) string on the console
- Write an EchoServer
  - return the received message to the client
- Use Writer/Reader class

Sizing Thread Pools

- Ideal size of a thread pool depends on
  - Types of tasks (I/O-bound, CPU-bound, …)
  - Characteristics of system (platform)

- Thread pool sizes should not be hard-coded
  - Should be adjusted adaptively to circumstances/configuration (processors, target utilization, …)
  - Runtime.availableProcessors

Sizing Thread Pools

- If thread pool size is too big
  - Threads compete for scarce CPU and memory resources, resulting in higher memory usage and possible resource exhaustion
- If thread pool size is too small
  - Throughput may suffer as processors remain unused despite available work
- Factors to consider
  - Computing environment, resource budget, nature of tasks
  - How many processors does the deployment system have?
  - How much memory?
  - Are tasks CPU-bound, I/O-bound or some combination?
  - Different categories of tasks?
    - Can have multiple thread pools, one for each category

Rule of Thumb

- Compute-intensive tasks
  - $N_{CPU}+1$ threads (N_{CPU}: number of CPUs)
  - Compute-intensive tasks can occasionally take a page fault or pause for some other reason, so another extra runnable thread prevents CPU cycles from going unused when this happens
- I/O-bound tasks or tasks that involve other blocking operations

$$N_{threads} = N_{CPU} \cdot U_{CPU} \cdot \left(1 + \frac{W}{C}\right)$$

U_{CPU} :	target CPU utilization
W :	waiting time
C :	compute time

Closing Server Sockets

- Closing a ServerSocket
  - release the port for other programs to use
  - break all currently open sockets that ServerSocket has accepted
- ServerSocket is closed automatically when a program dies

```
ServerSocket server = null;
try {
  server = new ServerSocket(port);
  // ... work with the server socket
} finally {
  if (server != null) {
    try {
      server.close();
    } catch (IOException ex) {
      // ignore
    }
```

```
ServerSocket server = new ServerSocket();
try {
  SocketAddress address = new InetSocketAddress(port);
  server.bind(address);
  // ... work with the server socket
} finally {
  try {
    server.close();
  } catch (IOException ex) {
    // ignore
  }
}
```

isClosed and isBound

- isClosed() `public boolean isClosed()`
  - return true if ServerSocket has been closed
  - ServerSocket object created with ServerSocket() and not yet bound to a port are not considered to be closed
- isBound() `public boolean isBound()`
  - return true if ServerSocket has ever been bound to a port, even if it's currently closed
- How do you test whether a ServerSocket is open?

Logging

Logging

- Servers run unattended for long period of time
  - need to record important info
- Two primary things to log
  - requests
  - server errors
- General rule of thumb
  - log only necessary, otherwise errors can be hidden due to too much log info
- Java package
  - java.util.logging
  - this is thread-safe

Creating and Using Logger

- Easiest to create one logger for each class

```
private final static Logger auditLogger = Logger.getLogger("requests");
```

name of log info

- Writing to a logger

```
catch (RuntimeException ex) {
  logger.log(Level.SEVERE, "unexpected error " + ex.getMessage(), ex);
}
```

- Seven levels defined in java.util.logging.Level
  - SEVERE(highest value), WARNING, INFO, CONFIG, FINE, FINER, FINEST(lowest value)

Example

```
import java.io.*;
import java.net.*;
import java.util.Date;
import java.util.concurrent.*;
import java.util.logging.*;

public class LoggingDaytimeServer {

  public final static int PORT = 13;
  private final static Logger auditLogger = Logger.getLogger("requests");
  private final static Logger errorLogger = Logger.getLogger("errors");

  public static void main(String[] args) {

   ExecutorService pool = Executors.newFixedThreadPool(50);

   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
         Socket connection = server.accept();
         Callable<Void> task = new DaytimeTask(connection);
         pool.submit(task);
       } catch (IOException ex) {
         errorLogger.log(Level.SEVERE, "accept error", ex);
       } catch (RuntimeException ex) {
         errorLogger.log(Level.SEVERE, "unexpected error " + ex.getMessage(), ex);
       }
     }
   } catch (IOException ex) {
     errorLogger.log(Level.SEVERE, "Couldn't start server", ex);
   } catch (RuntimeException ex) {
     errorLogger.log(Level.SEVERE, "Couldn't start server: " + ex.getMessage(), ex);
   }
  }
```

Example (contd.)

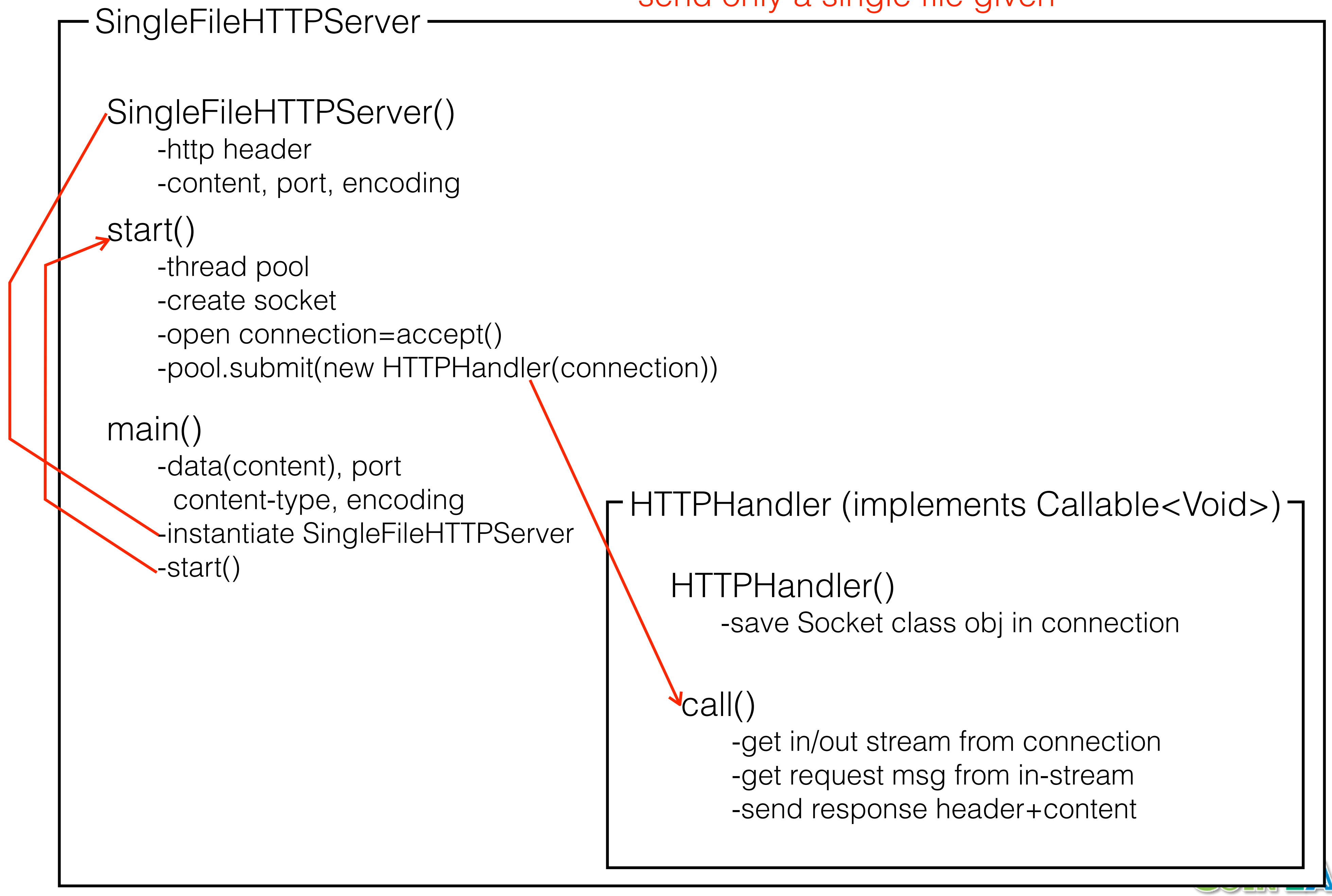
```java
private static class DaytimeTask implements Callable<Void> {

  private Socket connection;

  DaytimeTask(Socket connection) {
    this.connection = connection;
  }

  @Override
  public Void call() {
    try {
      Date now = new Date();
      // write the log entry first in case the client disconnects
      auditLogger.info(now + " " + connection.getRemoteSocketAddress());
      Writer out = new OutputStreamWriter(connection.getOutputStream());
      out.write(now.toString() +"\r\n");
      out.flush();
    } catch (IOException ex) {
        // client disconnected; ignore;
    } finally {
      try {
        connection.close();
      } catch (IOException ex) {
        // ignore;
      }
    }
    return null;
  }
 }
}
```

INFO level logging

Examples: HTTP Servers

Single-File Server

send only a single file given



SingleFileHTTPServer

SingleFileHTTPServer()
- -http header
- -content, port, encoding

start()
- -thread pool
- -create socket
- -open connection=accept()
- -pool.submit(new HTTPHandler(connection))

main()
- -data(content), port
  content-type, encoding
- -instantiate SingleFileHTTPServer
- -start()

HTTPHandler (implements Callable<Void>)

HTTPHandler()
- -save Socket class obj in connection

call()
- -get in/out stream from connection
- -get request msg from in-stream
- -send response header+content

```java
import java.io.*;
import java.net.*;
import java.nio.charset.Charset;
import java.nio.file.*;
import java.util.concurrent.*;
import java.util.logging.*;

public class SingleFileHTTPServer {

  private static final Logger logger = Logger.getLogger("SingleFileHTTPServer");

  private final byte[] content;
  private final byte[] header;
  private final int port;
  private final String encoding;

  public SingleFileHTTPServer(String data, String encoding,
      String mimeType, int port) throws UnsupportedEncodingException {
    this(data.getBytes(encoding), encoding, mimeType, port);
  }

  public SingleFileHTTPServer(
      byte[] data, String encoding, String mimeType, int port) {
    this.content = data;
    this.port = port;
    this.encoding = encoding;
    String header = "HTTP/1.0 200 OK\r\n"
        + "Server: OneFile 2.0\r\n"
        + "Content-length: " + this.content.length + "\r\n"
        + "Content-type: " + mimeType + "; charset=" + encoding + "\r\n\r\n";
    this.header = header.getBytes(Charset.forName("US-ASCII"));
  }
```

```java
public void start() {
  ExecutorService pool = Executors.newFixedThreadPool(100);
  try (ServerSocket server = new ServerSocket(this.port)) {
    logger.info("Accepting connections on port " + server.getLocalPort());
    logger.info("Data to be sent:");
    logger.info(new String(this.content, encoding));

    while (true) {
      try {
        Socket connection = server.accept();
        pool.submit(new HTTPHandler(connection));
      } catch (IOException ex) {
        logger.log(Level.WARNING, "Exception accepting connection", ex);
      } catch (RuntimeException ex) {
        logger.log(Level.SEVERE, "Unexpected error", ex);
      }
    }
  } catch (IOException ex) {
    logger.log(Level.SEVERE, "Could not start server", ex);
  }
}
```

```java
private class HTTPHandler implements Callable<Void> {
  private final Socket connection;

  HTTPHandler(Socket connection) {
    this.connection = connection;
  }

  @Override
  public Void call() throws IOException {
    try {
      OutputStream out = new BufferedOutputStream(
                              connection.getOutputStream()
                             );
      InputStream in = new BufferedInputStream(
                              connection.getInputStream()
                             );
      // read the first line only; that's all we need
      StringBuilder request = new StringBuilder(80);
      while (true) {
        int c = in.read();
        if (c == '\r' || c == '\n' || c == -1) break;
        request.append((char) c);
      }
      // If this is HTTP/1.0 or later send a MIME header
      if (request.toString().indexOf("HTTP/") != -1) {
        out.write(header);
      }
      out.write(content);
      out.flush();
    } catch (IOException ex) {
      logger.log(Level.WARNING, "Error writing to client", ex);
    } finally {
      connection.close();
    }
    return null;
  }
}
```

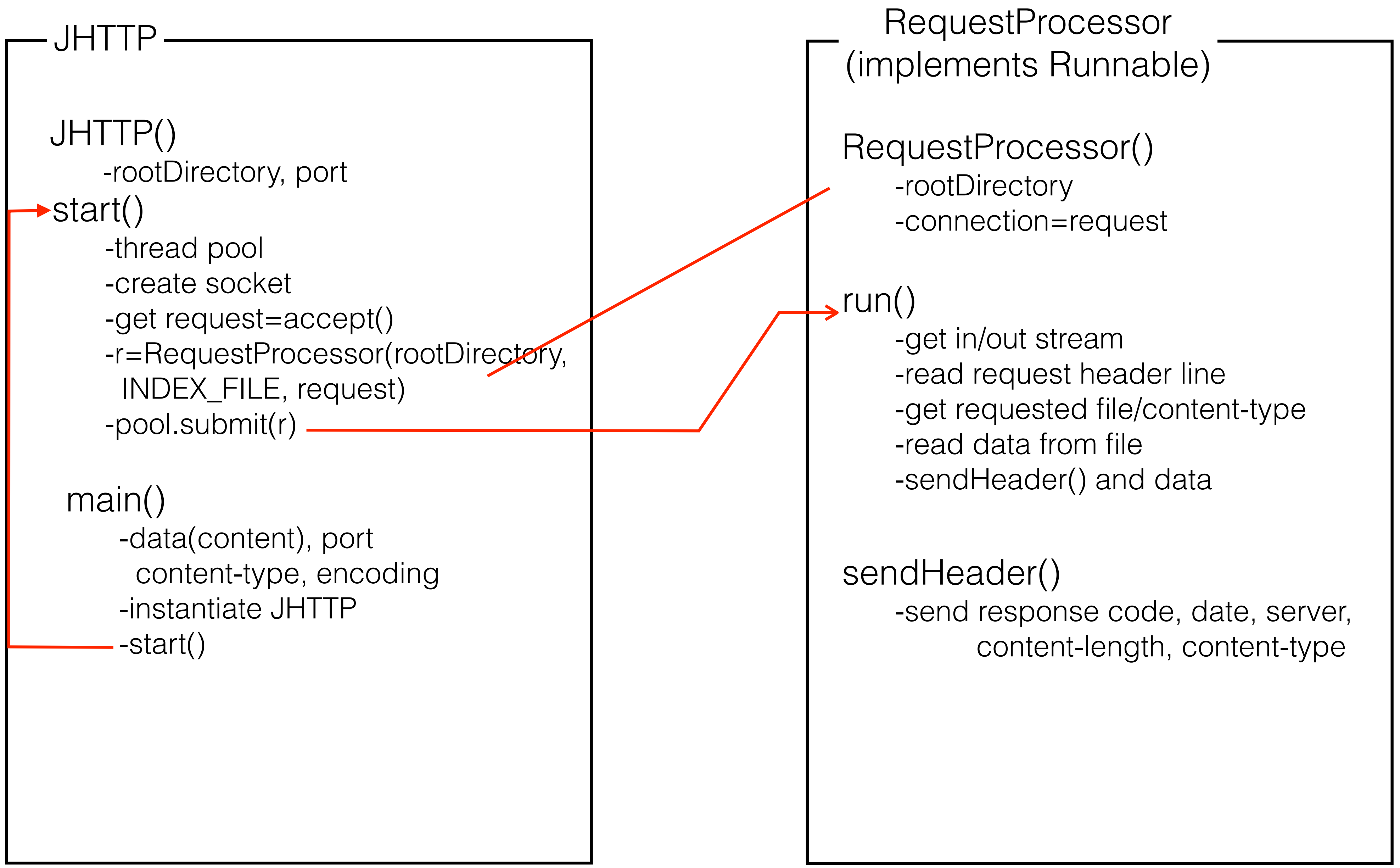
why?

```java
    public static void main(String[] args) {

      // set the port to listen on
      int port;
      try {
        port = Integer.parseInt(args[1]);
        if (port < 1 || port > 65535) port = 80;
      } catch (RuntimeException ex) {
        port = 80;
      }
      String encoding = "UTF-8";
      if (args.length > 2) encoding = args[2];

      try {
        Path path = Paths.get(args[0]);;
        byte[] data = Files.readAllBytes(path);

        String contentType = URLConnection.getFileNameMap().getContentTypeFor(args[0]);
        SingleFileHTTPServer server = new SingleFileHTTPServer(data, encoding,
            contentType, port);
        server.start();

      } catch (ArrayIndexOutOfBoundsException ex) {
        System.out.println(
            "Usage: java SingleFileHTTPServer filename port encoding");
      } catch (IOException ex) {
        logger.severe(ex.getMessage());
      }
    }
  }
```

JHTTP Web Server

- Send requested file if it exists
  - requested file is converted to a filename on the local file system
  - use canonical path to prevent sneaky client from walking all over the local file system by including ".." in URLs
  - if requested file is directory (name ends with slash), add the name of index file
  - include content-type
- Otherwise, send appropriate response code
  - 404 Not Found
  - 501 Not Implemented

Structure



JHTTP

JHTTP()
- -rootDirectory, port

start()
- -thread pool
- -create socket
- -get request=accept()
- -r=RequestProcessor(rootDirectory, INDEX_FILE, request)
- -pool.submit(r)

main()
- -data(content), port content-type, encoding
- -instantiate JHTTP
- -start()

RequestProcessor (implements Runnable)

RequestProcessor()
- -rootDirectory
- -connection=request

run()
- -get in/out stream
- -read request header line
- -get requested file/content-type
- -read data from file
- -sendHeader() and data

sendHeader()
- -send response code, date, server, content-length, content-type

```java
import java.io.*;
import java.net.*;
import java.util.concurrent.*;
import java.util.logging.*;

public class JHTTP {

  private static final Logger logger = Logger.getLogger(
      JHTTP.class.getCanonicalName());
  private static final int NUM_THREADS = 50;
  private static final String INDEX_FILE = "index.html";

  private final File rootDirectory;
  private final int port;

  public JHTTP(File rootDirectory, int port) throws IOException {

    if (!rootDirectory.isDirectory()) {
      throw new IOException(rootDirectory
          + " does not exist as a directory");
    }
    this.rootDirectory = rootDirectory;
    this.port = port;
  }
```

```java
public void start() throws IOException {
  ExecutorService pool = Executors.newFixedThreadPool(NUM_THREADS);
  try (ServerSocket server = new ServerSocket(port)) {
    logger.info("Accepting connections on port " + server.getLocalPort());
    logger.info("Document Root: " + rootDirectory);

    while (true) {
      try {
        Socket request = server.accept();
        Runnable r = new RequestProcessor(
            rootDirectory, INDEX_FILE, request);
        pool.submit(r);
      } catch (IOException ex) {
        logger.log(Level.WARNING, "Error accepting connection", ex);
      }
    }
  }
}
```

```
  public static void main(String[] args) {

    // get the Document root
    File docroot;
    try {
      docroot = new File(args[0]);
    } catch (ArrayIndexOutOfBoundsException ex) {
      System.out.println("Usage: java JHTTP docroot port");
      return;
    }

    // set the port to listen on
    int port;
    try {
      port = Integer.parseInt(args[1]);
      if (port < 0 || port > 65535) port = 80;
    } catch (RuntimeException ex) {
      port = 80;
    }

    try {
      JHTTP webserver = new JHTTP(docroot, port);
      webserver.start();
    } catch (IOException ex) {
      logger.log(Level.SEVERE, "Server could not start", ex);
    }
  }
}
```

```java
import java.io.*;
import java.net.*;
import java.nio.file.Files;
import java.util.*;
import java.util.logging.*;

public class RequestProcessor implements Runnable {

  private final static Logger logger = Logger.getLogger(
      RequestProcessor.class.getCanonicalName());

  private File rootDirectory;
  private String indexFileName = "index.html";
  private Socket connection;

  public RequestProcessor(File rootDirectory,
      String indexFileName, Socket connection) {

    if (rootDirectory.isFile()) {
      throw new IllegalArgumentException(
          "rootDirectory must be a directory, not a file");
    }
    try {
      rootDirectory = rootDirectory.getCanonicalFile();
    } catch (IOException ex) {
    }
    this.rootDirectory = rootDirectory;

    if (indexFileName != null) this.indexFileName = indexFileName;
    this.connection = connection;
  }
```

```java
@Override
public void run() {
  // for security checks
  String root = rootDirectory.getPath();
  try {
    OutputStream raw = new BufferedOutputStream(
                        connection.getOutputStream()
                       );
    Writer out = new OutputStreamWriter(raw);
    Reader in = new InputStreamReader(
                  new BufferedInputStream(
                   connection.getInputStream()
                  ),"US-ASCII"
                 );
    StringBuilder requestLine = new StringBuilder();
    while (true) {
      int c = in.read();
      if (c == '\r' || c == '\n') break;
      requestLine.append((char) c);
    }

    String get = requestLine.toString();

    logger.info(connection.getRemoteSocketAddress() + " " + get);
```

```
String[] tokens = get.split("\\s+");
String method = tokens[0];
String version = "";
if (method.equals("GET")) {
  String fileName = tokens[1];
  if (fileName.endsWith("/")) fileName += indexFileName;
  String contentType =
      URLConnection.getFileNameMap().getContentTypeFor(fileName);
  if (tokens.length > 2) {
    version = tokens[2];
  }

  File theFile = new File(rootDirectory,
      fileName.substring(1, fileName.length()));
  if (theFile.canRead()
      // Don't let clients outside the document root
      && theFile.getCanonicalPath().startsWith(root)) {
    byte[] theData = Files.readAllBytes(theFile.toPath());
    if (version.startsWith("HTTP/")) { // send a MIME header
      sendHeader(out, "HTTP/1.0 200 OK", contentType, theData.length);
    }

    // send the file; it may be an image or other binary data
    // so use the underlying output stream
    // instead of the writer
    raw.write(theData);
    raw.flush();
```

HTTP 1.0 or higher

```java
} else { // can't find the file
  String body = new StringBuilder("<HTML>\r\n")
      .append("<HEAD><TITLE>File Not Found</TITLE>\r\n")
      .append("</HEAD>\r\n")
      .append("<BODY>")
      .append("<H1>HTTP Error 404: File Not Found</H1>\r\n")
      .append("</BODY></HTML>\r\n").toString();
  if (version.startsWith("HTTP/")) { // send a MIME header
    sendHeader(out, "HTTP/1.0 404 File Not Found",
        "text/html; charset=utf-8", body.length());
  }
  out.write(body);
  out.flush();
}
```

```
    } else { // method does not equal "GET"
      String body = new StringBuilder("<HTML>\r\n")
          .append("<HEAD><TITLE>Not Implemented</TITLE>\r\n")
          .append("</HEAD>\r\n")
          .append("<BODY>")
          .append("<H1>HTTP Error 501: Not Implemented</H1>\r\n")
          .append("</BODY></HTML>\r\n").toString();
      if (version.startsWith("HTTP/")) { // send a MIME header
        sendHeader(out, "HTTP/1.0 501 Not Implemented",
                  "text/html; charset=utf-8", body.length());
      }
      out.write(body);
      out.flush();
    }
  } catch (IOException ex) {
    logger.log(Level.WARNING,
        "Error talking to " + connection.getRemoteSocketAddress(), ex);
  } finally {
    try {
      connection.close();
    }
    catch (IOException ex) {}
  }
}
```

```
  private void sendHeader(Writer out, String responseCode,
      String contentType, int length)
      throws IOException {
    out.write(responseCode + "\r\n");
    Date now = new Date();
    out.write("Date: " + now + "\r\n");
    out.write("Server: JHTTP 2.0\r\n");
    out.write("Content-length: " + length + "\r\n");
    out.write("Content-type: " + contentType + "\r\n\r\n");
    out.flush();
  }
}
```

Exercise

- Modify JHTTP.java/RequestProcessor.java to handle DELETE method
  - three cases
    - file not found (404)
    - file found (200)
      - file deleted => success
      - file not deleted => failure
  - useful methods
    - File.delete()
- Write a client that requests DELETE of a file on server
  - show server's response on console
  - use HttpURLConnection (that can set request method)
  - useful methods
    - HttpURLConnection.setRequestMethod()
    - HttpURLConnection.connect()