University of Genova

LM in Computer Engineering

SOFTWARE PLATFORM NOTES
Java Technology

Prof. Massimo Maresca

a.a. 2019-2020

ISBN: 979-12-200-5975-6

HTTP-SOCKET

The Http-Socket folder includes the following files:

- MyTcpServer.java
- MyTcpClient.java
- MyHttpServerOverTCP.java
- MyHttpServer.java + Handler.java
- GetTest.java

Description

1. MyTcpServer and MyTcpClient are two programs that connect over a Tcp socket.
2. MyHttpServer + Handler.java is a Http Server built on top of the Http library.
3. MyHttpServerOverTCP is an HTTP Server buit over a TCP socket. No Http Library is used. The Http behavior on the server side is obtained through a response message compliant with the Http standard.
4. GetTest is an Http client that can be used to test the servers.

Experiments

First compile the programs using the makeall batch file or directly the javac command on a Command Line Interface (CLI). Alternatively use the source code to build Exclipse projects.

1. Test the Tcp interaction between the client (MyTcpClient) and the server (MyTcpServer) over the locahost. Run the server on a CLI and the client on another CLI.
2. Test the Http interaction between a client (GetTest) and the server (MyHttpServer) over the local host using a direct Http connection. Run the server on a CLI and GetTest on another CLI. Alternatively run everything on Eclipse. The port number exposed by the server is 8500. Such a number must be put as an argument of the GetTest.
3. Test the Http interaction between a client (GetTest) and the server (MyHttpServerOverTCP) using a TCP socket. Run the server on a CLI and GetTest on another CLI. Alternatively run everything on Eclipse. The port number exposed by the server is 7654. Such a number must be put as an argument of the GetTest command.
4. Install the Apache Http Server from the Apache Web Site. Finalize its configuration and launch it. Test its presence through a local browser. Then run the client (GetTest).

The programs can be compiled through the makeall.bat script, which includes the following commands.

```
javac MyTcpServer.java
javac MyTcpClient.java
javac MyHttpServer.java
javac MyHttpServerOverTCP.java
javac GetTest.java
```

The source code of the programs is reported in the following pages.

MyTcpServer.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.*;
public class MyTcpServer  {
    public static void main(String [] args) throws IOException {
        ServerSocket sSoc = new ServerSocket(8500);
        Socket s = sSoc.accept();
        System.out.println("After Accept");
        PrintStream out =
            new PrintStream(s.getOutputStream());
        try {
            Thread.sleep(4000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String strOut = new String ("Hi, I am the Server");
        out.println(strOut);
        System.out.println("After sending to client");
        out.flush();
        System.out.println("After flush");
        try {
            Thread.sleep(4000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        sSoc.close();
    }

}
```

MyTcpClient.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;


 public class MyTcpClient  {

    public static void main(String [] args) {
        Socket s;
        try {
            s = new Socket("127.0.0.1", 8500);
            BufferedReader in = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
            String strIn = in.readLine();
            System.out.println(strIn);
        } catch (IOException e) {
            System.out.println("Error in Socket");
            System.exit(-1);
        }
    }

}
```

MyHttpServerOverTCP.java

```
import java.io.*;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.ByteBuffer;
public class MyHttpServerOverTCP {
    public static void main(String[] args) {
        MyHttpServerOverTCP server = new MyHttpServerOverTCP();
        server.await();
    }
    public void await() {
        String welcomeString =     "HTTP/1.1 200 OK\r\n"+"Content-Type: text/html\r\n"+"Content-Length:
22\r\n" +"\r\n"+"<h1>Here I am!!!!</h1>";
        ServerSocket serverSocket = null; int port = 7654;
        try {
            serverSocket = new ServerSocket(port, 1, InetAddress.getByName("127.0.0.1"));
        }
        catch (IOException e) {
            e.printStackTrace(); System.exit(1);
        }
        Socket socket = null;
        InputStream input = null; OutputStream output = null;
        try {
                socket = serverSocket.accept();
                input = socket.getInputStream();
                output = socket.getOutputStream();
                output.write(welcomeString.getBytes());
                System.out.println("Connection accepted from " + socket.getInetAddress());
        }
            catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

MyHttpServer.java

```
import com.sun.net.httpserver.HttpContext;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
import com.sun.net.httpserver.HttpServer;
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;
import java.nio.charset.StandardCharsets;
import com.sun.net.httpserver.HttpServer;
public class MyHttpServer {
    public static void main(String[] args) throws IOException {
        MyHttpServer myHttpSever = new MyHttpServer();
    }
    MyHttpServer() throws IOException {
        HttpServer server = HttpServer.create(new InetSocketAddress(8500), 0);
        HttpContext context = server.createContext("/");
        Handler handler = new Handler();
        context.setHandler(handler);
        server.start();
    }
}
```

Handler.java

```
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import com.sun.net.httpserver.HttpExchange;
import com.sun.net.httpserver.HttpHandler;
public class Handler implements HttpHandler {
    String RESPONSE = "<html><body><p>Hello World!</body></html>";
      @Override
    public void handle(HttpExchange exchange) throws IOException {
        exchange.getResponseHeaders().add("encoding", "UTF-8");
        exchange.sendResponseHeaders(200, RESPONSE.length());
        exchange.getResponseBody().write(RESPONSE.getBytes(StandardCharsets.UTF_8));
        exchange.close();
      }
}
```

GetTest.java

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
 public class GetTest {
    public static void main(String[] args) throws Exception {
        GetTest http = new GetTest();
        System.out.println("Testing 1 - Send Http GET request");
        http.sendGet(args);
    }
    private void sendGet(String[] args) throws Exception {
        String url = "";
        if (args.length==0) url = "http://127.0.0.1:80";
        else {
            System.out.println(args[0]);
            url = "http://127.0.0.1:" + args[0];
        }
        System.out.println("Connecting to: "+ url);
        URL obj = new URL(url);
        HttpURLConnection con = (HttpURLConnection) obj.openConnection();
        con.setRequestMethod("GET");
        int responseCode = con.getResponseCode();
        System.out.println("\nSending 'GET' request to URL : " + url);
        System.out.println("Response Code : " + responseCode);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();
// Doing Client job
         System.out.println("Response: " + response.toString());
    }
}
```

THREADS

The Threads folder includes two service folders:

- Lib: A directory containing the Command Line parsing library
- Number of CPUs: A directory containing a program that gives the number of CPUs as a result

It also includes seven folders containing the following programs:

1. ASimpleThread: A program that creates and initializes five Threads and run them concurrently.
2. SingleThreadedServer: A computing intensive program that does not use threads. It just executes the task sequentially
3. ThreadPerTaskServer: A computing intensive program that uses an unlimited number of threads
4. ThreadPool: A computing intensive program that uses a configurable thread pool
5. ThreadPoolImplementation: A program showing the way a Thread Pool works.
6. SocketWithThread: A program to test the combination of sockets and threads.
7. SocketWithThreadAndPool: A program to test the combination of sockets, threads and thread pools.

The next pages present the source code of the programs.

```
public class ThreadMain {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
                System.out.println("Generating Thread n. "+ i);
                MyThread T= new MyThread(i);
                T.start();
            }
    }
}

public class MyThread extends Thread {
    int n;
    MyThread(int i){
        n=i;
    }

    public void run (){
        String threadName = Thread.currentThread().getName();

        for (int i=0; i < 5; i++) {
            try {
                Thread.currentThread().sleep(1000);
                int val = n*10+i;
                System.out.println(threadName + " " + val);
            } catch (Exception e) {
                System.out.println(e.toString());
            }
        }
    }
}
```

SingleThreadedServer

The SingleThreadedServer folder includes two classes, namely STSMain.java, which is the main program, and SingleThreadedServer.java, which is the actual program.
The makeSTSMain.bat batch command can be used to compile the programs using a command line. No parameter is required.
To execute the program use the following command:
java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. STSMain <opzioni>
Options:
-n number of tasks
-l number of iterations inside tasks (computational load)
-v verbose (to visualize the detail of each iteration)
The default options values are:
-n 100
-l 1000000

A non verbose run gives as a result: Number of tasks - Total elapsed time - Time per Task

Here is some result:

1. Qui c'e' solo processing (l = 1000000)
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. STSMain -n 10
   10 3247 324.7
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. STSMain -n 20
   20 6467 323.35

STSMain.java

```
import org.apache.commons.cli.*;
@SuppressWarnings( "deprecation" )
public class STSMain {
    public static void main(String[] args) throws Exception {
        int n = 100, l = 1000000, p = 10, w = 0;
        boolean v = false;
        Options options = new Options();
        options.addOption("n", true, "Number of Tasks");
        options.addOption("l", true, "Task processing intensity (number of iterations)");
        options.addOption("p", true, "Thread pool size");
        options.addOption("w", true, "Task waiting intensity (Wait time)");
        options.addOption ("v", false, "verbose");
        CommandLineParser parser = new DefaultParser();
        try {
            CommandLine cmd = parser.parse(options, args);
            if(cmd.hasOption("n")) n=Integer.parseInt(cmd.getOptionValue ("n"));
            if(cmd.hasOption("l")) l=Integer.parseInt(cmd.getOptionValue ("l"));
            if(cmd.hasOption("v")) v=true;
            if(cmd.hasOption("h")) {
                System.out.println("Usage: " + "-n number of tasks -l task length -v verbose");
                System.exit(0);
            }
        } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
        SingleThreadedServer s = new SingleThreadedServer();
        long m1 = System.currentTimeMillis();
        s.go(n, l, v);
        long m2 = System.currentTimeMillis() - m1;
        System.out.println(n+" "+m2+" "+ ((float) m2/ (float) n));
    }
}
```

SingleThreadedServer.java

```
import java.io.IOException;
import java.lang.System;
import java.util.Random;
class SingleThreadedServer{
    long m1, m2;
    void handleRequest(int iterationNumber, int fakeparam, int timeWasteIterationNumber, boolean v) {
        long startTaskTime=System.currentTimeMillis();
        for (int i=0; i < timeWasteIterationNumber; i++) {
            for (int j=0; j<1000; j++) {
                if ((i*j) == fakeparam) {
                    System.out.println(i);
                }
            }
        }
        long endTaskTime=System.currentTimeMillis();
        if (v) {
            System.out.print("Task n. "+ iterationNumber + " completed at time: "+ (endTaskTime-m1) +
                        " started at time: " + (startTaskTime-m1));
            System.out.println(" Elapsed time:("+ (endTaskTime-startTaskTime)+")");

    }
    void go(int n, int load, boolean v){
        m1 = System.currentTimeMillis();
        for (int i = 0; i < n; i++){
            m2 = System.currentTimeMillis()-m1;
            if (v)
                System.out.println("Task: " + i + " submitted at "+ m2);
            handleRequest(i, -1, load, v);
        }
    }
}
```

ThreadPerTaskServer

The ThreadPerTaskServer folder includes two classes as follows:

- TPTMain.java includes the main program
- ThreadPerTask.java includes the actual program to be tested

makeTPTMain.bat is a batch file that takes care of compilation using a command line. It does not require parameters.

Run the program through the following comman.

java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. TPTMain <opzioni>
The options are:
-n number of tasks
-l number of iterations in tasks (computational load)
-v verbose to display the details during execution
The default option values are:
-n 100
-l 1000000

The result of each run is as follows: Number of tasks - Total Elapsed Time – Time per Task

In the following some results:

java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. TPTMain -n 50
50 4420 88.4
java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. TPTMain -n 100
100 8798 87.98

```
import org.apache.commons.cli.*;
@SuppressWarnings( "deprecation" )
public class TPTMain {
    public static void main(String[] args) throws Exception {
        int n = 100;
        int l = 1000000;
        boolean v = false;

        Options options = new Options();
        options.addOption("n", true, "Number of Tasks");
        options.addOption("l", true, "Task processing intensity (number of iterations)");
        options.addOption ("v", false, "verbose");
        options.addOption ("h", false, "help");


        CommandLineParser parser = new DefaultParser();

        try {
            CommandLine cmd = parser.parse(options, args);
            if(cmd.hasOption("n")) n=Integer.parseInt(cmd.getOptionValue ("n"));
            if(cmd.hasOption("l")) l=Integer.parseInt(cmd.getOptionValue ("l"));
            if(cmd.hasOption("v")) v=true;
            if(cmd.hasOption("h")) {
                System.out.println("Usage: " + "-n number of tasks -l task length -v verbose");
                System.exit(0);
            }
        } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
        ThreadPerTaskServer s = new ThreadPerTaskServer();
        long m1 = System.currentTimeMillis();
        s.go(n, l, v);
        long m2 = System.currentTimeMillis() - m1;
        System.out.println(n+" "+m2+" "+ ((float) m2/ (float) n));
    }
}
```

ThreadPerTaskServer.java

```
import java.io.IOException;
import java.lang.System;
import java.util.Random;
import java.util.*;

class ThreadPerTaskServer{
    long m1, m2;
    void handleRequest(int taskNumber, int fakeparam, int timeWasteIterationNumber, boolean v) {
        long startTaskTime=System.currentTimeMillis();
        for (int i=0; i < timeWasteIterationNumber; i++) {
            for (int j=0; j<1000; j++) {
                if ((i*j) == fakeparam) {
                    System.out.println(i);
                }
            }
        }
        long endTaskTime=System.currentTimeMillis();
        if (v)
            System.out.println("Task n. "+ taskNumber + " completed at time: "+ (endTaskTime-m1) +
                               " started at time: " + (startTaskTime-m1) +
                               " Elapsed time:("+ (endTaskTime-startTaskTime)+")");
    }

    public class HandleRequestThread implements Runnable {
      private int  i;
      private int l;
      private boolean v;
      public HandleRequestThread(int i, int l, boolean v) {
         this.i = i;
         this.l = l;
         this.v = v;
      }
      public void run() {
          handleRequest(i, -1, l, v);
      }
    }

    void go(int n, int load, boolean v){
        Thread[] threads = new Thread[n];
        List<Thread> allThreads = new ArrayList<Thread>();
```

```
        for(Thread thread : threads){
            if(null != thread){
              if(thread.isAlive()){
                    allThreads.add(thread);
              }
            }
        }
        for (int i=0; i < n ; i++) {
            threads[i]= new Thread(new HandleRequestThread(i, load, v));
            threads[i].start();
        }
        try {
            for (Thread thread : threads) {
                thread.join();
            }
        } catch (InterruptedException e) {
            System.out.println("Error");;
            System.exit(-1);
        }
    }
}
```

ThreadPool

The ThreadPool folder includes two classes as follows:

- MTSMain.java includes the main program
- ThreadPoolServer.java includes the actual program under test

makeMTSMain.bat is a batch file that can be used to compile the programs, It does not requires parameters.

Run the program through the following command:
java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain <opzioni>
The options are:
-n number of tasks
-p number of threads in the pool
-l number of iterations in tasks (Computational load)
-w wait time in tasks
-v verbose
The default values are:
-n 100
-p 10
-l 1000000
-w 0

Program runs in non verbose mode produces the following results:
Number of tasks – Number of Threads in the Pool – Total elapsed time – Time per Task

Now some results:

2. Only processing (l = 1000000) and just 1 Thread
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 10 -p 1
   10 1 3287 0 328.7
3. Processing (l = 1000000) + Wait di 300 msec and just q Thread
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 10 -p 1 -w 300
   10 1 6566 300 656.6

Let us now look at some more complex run:

1. Only processing (l = 1000000) and 1, 2, 4, 8, 16 Threads
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -p 1
   50 1 16532 0 330.64
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -p 2
   50 2 8625 0 172.5
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -p 4
   50 4 5536 0 110.72
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -p 8
   50 8 4686 0 93.72
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -p 16
   50 16 4545 0 90.9

N.B. We now raise the number of tasks to keep the Threads busy ….

   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 100 -p 32
   100 32 9015 0 90.15

2. Processing (l = 1000000) + a 300 msec wait and again 1, 2, 4, 8, 16 Threads
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -w 300 -p 1
   50 1 32916 300 658.32
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -w 300 -p 2
   50 2 16456 300 329.12
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -w 300 -p 4
   50 4 9080 300 181.6
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -w 300 -p 8
   50 8 6481 300 129.62
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 50 -w 300 -p 16
   50 16 5739 300 114.78

N.B. We now raise the number of tasks to keep the Threads busy ….

   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 100 -w 300 -p 32
   100 32 9901 300 99.01
   java -cp ..\lib\commons-cli-1.4\commons-cli-1.4.jar;. MTSMain -n 200 -w 300 -p 64
   200 64 19314 300 96.57

It appears that the wait periods take advantage of multithreadung. In absence of waits the program converges to 90 with 16 threads and then does not improve.

```
import org.apache.commons.cli.*;
@SuppressWarnings( "deprecation" )
public class MTSMain {
    public static void main(String[] args) throws Exception {
        int n = 100;
        int l = 1000000;
        int p = 10;
        int w = 0;
        boolean v = false;

        Options options = new Options();
        options.addOption("n", true, "Number of Tasks");
        options.addOption("l", true, "Task processing intensity (number of iterations)");
        options.addOption("p", true, "Thread pool size");
        options.addOption("w", true, "Task waiting intensity (Wait time)");
        options.addOption ("v", false, "verbose");

        CommandLineParser parser = new DefaultParser();

        try {
            CommandLine cmd = parser.parse(options, args);
            if(cmd.hasOption("n")) {
                n=Integer.parseInt(cmd.getOptionValue ("n"));
            }
            if(cmd.hasOption("l")) {
                l=Integer.parseInt(cmd.getOptionValue ("l"));
            }
            if(cmd.hasOption("p")) {
                p=Integer.parseInt(cmd.getOptionValue ("p"));
            }
            if(cmd.hasOption("w")) {
                w=Integer.parseInt(cmd.getOptionValue ("w"));
            }

            if(cmd.hasOption("v")) {
                v=true;
            }
            if(cmd.hasOption("h")) {
```

```
                System.out.println("Usage: " + "-n number of tasks -l task length -p thread pool size -w
wait time -v verbose");
                System.exit(0);
            }
        } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }

        ThreadPoolServer s = new ThreadPoolServer();
        long m1 = System.currentTimeMillis();
        s.go(n, l, p, w, v);
        long m2 = System.currentTimeMillis() - m1;
        System.out.println(n+" "+p+" "+m2+" "+w+" "+ ((float) m2/ (float) n));

    }
}
```

```
import java.io.IOException;
import java.lang.System;
import java.util.Random;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

class ThreadPoolServer{
    long m1, m2;
    void handleRequest(int iterationNumber, int fakeparam, int l, int w, boolean v) {
        long startTaskTime=System.currentTimeMillis();
        for (int i=0; i < l; i++) {
            for (int j=0; j<1000; j++) {
                if ((i*j) == fakeparam) {
                    System.out.println(i);
                }
            }
        }
        try {
            Thread.sleep(w);
        } catch (Exception e) {
            System.out.println("Errror in sleep");
            System.exit(-1);
        }
        long endTaskTime=System.currentTimeMillis();
        if (v)
            System.out.println("Task n. "+ iterationNumber + " completed at time: "+ (endTaskTime-m1) +
                        " started at time: " + (startTaskTime-m1) + " Elapsed time:("+ (endTaskTime-
startTaskTime)+")");
    }

    class HandleRequestThread implements Runnable {
      private int  i;
      private int l;
      private int w;
```

```
      private boolean v;
      public HandleRequestThread(int i, int l, int w, boolean v) {
         this.i = i;
         this.l = l;
         this.v = v;
         this.w = w;
      }
      public void run() {
//          Random rand = new Random();
//          handleRequest(i, -1, rand.nextInt(l), v);
          handleRequest(i, -1, l, w, v);
      }
    }

    void go(int n, int load, int p, int w, boolean v){
//        Random rand = new Random();
        m1 = System.currentTimeMillis();
        ExecutorService pool = Executors.newFixedThreadPool(p);
        for (int i = 0; i < n; i++){
            m2 = System.currentTimeMillis() - m1;
            Runnable h = new  HandleRequestThread(i, load, w, v);
            if (v)
                System.out.println("Task: " + i + " submitted at "+ m2);
            pool.execute(h);
        }
        pool.shutdown();
        try {
            if (!pool.awaitTermination(60, TimeUnit.SECONDS)) {
                pool.shutdownNow();
            }
        } catch (InterruptedException ex) {
            pool.shutdownNow();
            Thread.currentThread().interrupt();
        }
    }
}
```

ThreadPoolImplementation

The ThreadPoolImplementation folder includes four programs that show the way a Thread Pool works.

The program includes the generation of 20 **tasks** and their insertion in a FIFO queue.
Each task has a configurable duration.
Then a set of **workers** of configurable size, extract the task from the queue and process them.
Here is the list of programs:
Service.java is the main program, which configures the system as follows:
-i time between consecutive task insertions in the queue
-d task duration
-p number of workers (pool size)

Examples of interesting command line options:
-p 20 -i 50 -d 10000
-p 10 -i 50 -d 10000
-p 10 -i 500 -d 1000

```
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

import org.apache.commons.cli.*;

@SuppressWarnings( "deprecation" )
public class Service {
    public static long inittime;
    public static int nWorkers;
    public static int taskDuration;
    public static int taskIntroductionDelay;

    @SuppressWarnings( "deprecation" )
    public static void main (String[] args) {
        Options options = new Options();
        options.addOption("p", true, "Thread Pool Size2");
        options.addOption("d", true, "Task Duration");
        options.addOption("i", true, "Task Introduction Delay");
        CommandLineParser parser = new DefaultParser();
        int nWorkers = 5;
        taskDuration = 2500;
        taskIntroductionDelay = 1000;
        try {
            CommandLine cmd = parser.parse(options, args);
            if(cmd.hasOption("p")) {
                nWorkers=Integer.parseInt(cmd.getOptionValue ("p"));
            }
            if(cmd.hasOption("d")) {
                taskDuration = Integer.parseInt(cmd.getOptionValue ("d"));
            }
            if(cmd.hasOption("i")) {
                taskIntroductionDelay = Integer.parseInt(cmd.getOptionValue ("i"));
            }
        } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
```

```
        System.out.println ("-p " + nWorkers + " -d " + taskDuration + " -i " + taskIntroductionDelay);
        inittime = System.currentTimeMillis();
        BlockingQueue<Task> queue = new ArrayBlockingQueue<>(10);
        TaskSource taskSource = new TaskSource (queue);
        Worker worker = new Worker (queue);

        for (int i = 0; i < nWorkers; i++) {
            new Thread(worker).start();
        }

        new Thread(taskSource).start();

    }

}
```

Task.java

```
public class Task implements Runnable {
    private String name;

    public Task(String str){
        this.name=str;
    }

    public String getName() {
        return name;
    }
    public void run() {
            System.out.println("Time: "+ (System.currentTimeMillis()-Service.inittime) + ": "+
                               name + " in Worker " + Thread.currentThread().getName() + " Starting");
        try {
            Thread.sleep(Service.taskDuration);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
            System.out.println("Time: "+ (System.currentTimeMillis()-Service.inittime) + ": "+
                               name + " in Worker " + Thread.currentThread().getName() + " Done");
    }
}
```

TaskSource.java

```
import java.util.concurrent.BlockingQueue;

public class TaskSource implements Runnable {

    private BlockingQueue<Task> queue;

    public TaskSource(BlockingQueue<Task> q){
        this.queue=q;
    }
    @Override
    public void run() {
        for(int i=0; i<20; i++){
            Task task = new Task ("Task n. "+i);
            try {
                System.out.println("Time: "+ (System.currentTimeMillis()-Service.inittime) + ": "+
"Enqueueing "+task.getName());
                queue.put(task);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            try {
                Thread.sleep(Service.taskIntroductionDelay);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Worker.java

```
import java.util.concurrent.BlockingQueue;

public class Worker implements Runnable{

private BlockingQueue<Task> queue;
    public Worker (BlockingQueue<Task> q){
        this.queue=q;
    }

    @Override
    public void run() {
        try{
            Task task;
            while(true) {
                task = queue.take();
                task.run();
            }
        }catch(InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

SocketWithThreads

The SocketWithThreads folder includes two programs, namely:

- CSocketWithThreads.java
- RSocketWithThreads.java

CSocketWithThreads is a simple client to test RSocketWithThreads
RSocketWIthThreads creates a socket and places itself in listen mode.
Then at each connection request (accept()) it generates a Thread that periodically writes a message on the local console

RSocketWithThread.java

```
import java.net.ServerSocket;
import java.net.Socket;


public class RSocketWithThreads {
    public static void main(String[] args) {
        int i = 0;
        try {
            ServerSocket sSoc = new ServerSocket(9000);
            while (true){
                Socket inSoc = sSoc.accept();
                System.out.println("Generating Thread n. "+ i);
                MyThread T= new MyThread(inSoc);
                T.start();
                i++;
            }
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

MyThread.java

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


public class MyThread extends Thread {
    private Socket localThreadSocket;
    static int n=0;
    MyThread(Socket s){
        localThreadSocket=s;
    }
    public void run (){
//        String s = Thread.currentThread().toString();
        String threadName = Thread.currentThread().getName();
        try{
            PrintStream out = new PrintStream(localThreadSocket.getOutputStream());
            out.println(threadName);
            out.close();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
        n++;

        for (int i=0; i < 10; i++) {
            try {
                Thread.currentThread().sleep(1000);
                System.out.println(threadName + " " + i);
            } catch (IOError | InterruptedException  e) {
                System.out.println(e.toString());
            }
        }
    }
}
```

CSocketWithThreads.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.Socket;


public class CSocketWithThreads {

    public static void main(String[] args) {
        try
        {
            Socket s = new Socket("localhost", 9000);
            BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            System.out.println(in.readLine());
            s.close();
        } catch (Exception e) {
            System.out.println("Segnalazione di errore: " + e.toString());
        }
    }
}
```

SocketWithThreadAndPool

The SocketWithThreadAndPool folder contains two programs, namely:

- CSocketWithThreads.java
- RSocketWithThreads.java

CSocketWithThreads is a simple client to test RSocketWithThreads
RSocketWIthThreads creates a socket and places itself in listen mode.
Then at each connection request (accept()) it generates a Thread that periodically writes a message on the local console and submit the Thread to a Thread Pool. The size of the Thread Pool must be given in the Command Line.

RSocketWithThreads.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.lang.*;

public class RSocketWithThreads {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println ("command <pool size>");
            System.exit(0);
        }
        int poolSize = Integer.parseInt(args[0]);
        ExecutorService pool = Executors.newFixedThreadPool(poolSize);
        try {
            ServerSocket sSoc = new ServerSocket(9000);
            int i = 0;
            while (true){
                Socket inSoc = sSoc.accept();
                System.out.println("Generating Thread n. "+ i);
                MyThread T= new MyThread(inSoc, i);
                pool.execute(T);
//                T.start();
                i++;
            }
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

MyThread.java

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

public class MyThread extends Thread {
    private Socket localThreadSocket;
        static int n;

    MyThread(Socket s, int n){
        this.n = n;
        localThreadSocket=s;
    }
    public void run (){
//        String s = Thread.currentThread().toString();
        String threadName = Thread.currentThread().getName();
        try{
            PrintStream out = new PrintStream(localThreadSocket.getOutputStream());
            out.println(threadName);
            out.close();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
        n++;

        for (int i=0; i < 5; i++) {
            try {
                Thread.currentThread().sleep(1000);
                System.out.println(threadName + " " + n + " " + i);
            } catch (IOError | InterruptedException  e) {
                System.out.println(e.toString());
            }
        }
    }
}
```

CSocketWithThreads.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.Socket;


public class CSocketWithThreads {

    public static void main(String[] args) {
        try
        {
            Socket s = new Socket("localhost", 9000);
            BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));
            System.out.println(in.readLine());
            s.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
}
```

Asnchronous IO-1

The Asnchronous IO-1 directory includes two programs, namely MyServer.java, a regular server which has nothing to do with NIO, and MyNIOClient.java, a NIO based client showing the way NIO works.

You can compile the two programs through the following commands:

```
javac MyServer.java
javac MyNIOClient.java
```

The server places itself in wait over port 8832, accepts the connection immediately (this is not under our control), waits 4 seconds, sends a message ("Hi I am the server") to the client, waits other 4 seconds and closes the connection.

The client does an asynchronous "connect" and then an asynchronous read. More specifically, it creates a SocketChannel, configures it as a non-blocking element, creates a Selector, activates it over all valid operations, and associates the SocketChannel to the Selector. Then it does the asynchronous connect and immediately prints the result which is typically negative (see the attached printout). In other words, in spite of the fact that the connection is local (to localhost) at the print time the connect function is still pending. Finally the program enters a while (true) loop (which keeps the CPU busy), and wait for the "connectable" event, which signals connection acceptance, from the server, and later the "readable" event, signaling that some data has arrived and is ready to be read.

Upon the arrival of the "connectable" event the client finalizes the connection whereas upon the arrival of the "readable" event the client reads and prints the data

All printed messages include the indication of the time at which they have taken place. Time is displayed as a difference with respect to the program start.

In the following an example of a printout of a run on localhost.

```
367:Not immediately Connected
368:Connectable
371:Finishing Connection...
373:finishConnect positive
2368:Readable
2368:byteRead = 21
2377:Hi, I am the Server
4372:Readable
4373:byteRead = 35
4375:Hi, this is the second message...
4377:Readable
4377:byteRead = 34
4379:* signals end of transmission...
Exiting...
```

MyServer.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.*;

public class MyServer  {
    public static void main(String [] args) throws IOException {
        ServerSocket sSoc = new ServerSocket(8832);
        Socket s = sSoc.accept();
        System.out.println("After Accept");
        PrintStream out = new PrintStream(s.getOutputStream());
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String strOut0 = new String ("Hi, I am the Server");
        out.print(strOut0);
        System.out.println("After sending to client");
        out.flush();
        System.out.println("After flush");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String strOut1 = new String ("Hi, this is the second message...");
        out.print(strOut1);
        System.out.println("After sending to client");
        out.flush();
        System.out.println("After flush");

        String strOut2 = new String ("* signals end of transmission...");
        out.print(strOut2);
        System.out.println("After sending to client");
        out.flush();
        System.out.println("After flush");
        try {
            Thread.sleep(2000);
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        sSoc.close();
    }
}
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.util.Iterator;
import java.util.Set;

public class MyNIOClient {
    static long m1;

    public static void main (String args[]) throws IOException {
        int readyChannels;
        int bytesRead;

        m1 = System.currentTimeMillis();

        SocketChannel socketChannel = SocketChannel.open();
        socketChannel.configureBlocking(false);
        Selector selector = Selector.open();

        int ops = socketChannel.validOps();
        SelectionKey key = socketChannel.register(selector, ops);

        InetSocketAddress myAddr = new InetSocketAddress("localhost", 8832);
        boolean connected = socketChannel.connect (myAddr);
        if (connected) {
            log ("Immediately Connected");
        } else {
            log ("Not immediately Connected");
        }
        while (true) {
            readyChannels = selector.select();
            if (readyChannels > 0 ) {
                if (key.isValid()) {}
                else {
```

```
                    socketChannel.close();
                }
                if (key.isConnectable()) {
                    log ("Connectable");
                    log ("Finishing Connection...");
                    boolean finished = socketChannel.finishConnect();
                    if (finished) {
                        log ("finishConnect positive");
                    } else {
                        log ("finishConnect negative");
                    }
                }
                if (key.isReadable()) {
                    log ("Readable");
                    ByteBuffer buf = ByteBuffer.allocate(1000);
                    bytesRead = socketChannel.read(buf);
                    log ("byteRead = " + bytesRead);
                    String s = new String (buf.array(), 0, bytesRead);
                    log (s);
                    if (s.contains("*")) {
                        System.out.println("Exiting...");
                        System.exit(0);
                    }
                }
            }
        }
    }
    static void log (String s) {
        long current_relative_time = System.currentTimeMillis()-m1;
        System.out.println(current_relative_time + ":" + s);
    }
}
```

Asynchronous IO-2

The Asynchronous IO-2 folder includes two programs, namely MyServer0.java, a regular server that has nothing to do with NIO, and ExampleNIO, an Eclipse project that acts as a client.
The client project must run after the activation of the server.
It activates a NIO platform and two threads that manage connections.
You can compile the server through the following command:
javac MyServer0.java

You can compile and run the client as an Eclipse project.

MyServer0.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.*;

public class MyServer0  {

    public static void main(String [] args) throws IOException {

        try {
            ServerSocket sSoc = new ServerSocket(8832);
            int i = 0;
            while (true){
                Socket inSoc = sSoc.accept();
                System.out.println("Generating Thread n. "+ i);
                MyThread0 T= new MyThread0(inSoc, i);
                T.start();
                i++;
            }
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```

MyThread0.java

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


public class MyThread0 extends Thread {
    private Socket localThreadSocket;
    static int n;
    PrintStream out;

    MyThread0(Socket s, int n){
        this.n = n;
        localThreadSocket=s;
    }
    public void run (){
        System.out.println("After Accept");
        try {
            out = new PrintStream(localThreadSocket.getOutputStream());
        } catch (Exception e) {
            e.printStackTrace();
        }try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String strOut0 = new String ("Hi, I am the Server");
        out.print(strOut0);
        System.out.println("After sending to client");
        out.flush();
        System.out.println("After flush");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String strOut1 = new String ("Hi, this is the second message...");
        out.print(strOut1);
        System.out.println("After sending to client");
        out.flush();
```

```
            System.out.println("After flush");

            String strOut2 = new String ("* signals end of transmission...");
            out.print(strOut2);
            System.out.println("After sending to client");
            out.flush();
            System.out.println("After flush");
            try {
                Thread.sleep(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            try {
                localThreadSocket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

The Eclipse Project source files – Program.java

```
import java.io.IOException;
import java.nio.channels.SelectionKey;
import java.nio.channels.SocketChannel;
import java.util.LinkedList;
import java.util.Queue;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class Program {
    public static long initialTime;
    public static Platform platform;
    public static Queue<Client> registrationQueue;
    public static Queue<SocketChannel> socketCloseQueue;

    static void sleep(long i) {
        try {
            Thread.sleep(i);
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }

    }
    public static void main(String[] args) {
        initialTime = System.currentTimeMillis();

        registrationQueue = new LinkedList<Client>();
        socketCloseQueue = new LinkedList<SocketChannel>();

        Platform platform = new Platform();
        try {
            platform.start();
        } catch (Exception e) {
            e.printStackTrace();
        }


        Client c0 = new Client("Client A");
        Client c1 = new Client("Client B");
        Client c2 = new Client("Client C");
        Client c3 = new Client("Client D");
```

```
        Client c4 = new Client("Client E");

        try {
            c0.doConnect();
            sleep(200);
            c1.doConnect();
            sleep(200);
            c2.doConnect();
            sleep(200);
            c3.doConnect();
            sleep(200);
            c4.doConnect();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.ClosedChannelException;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.SocketChannel;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Queue;
import java.util.Set;


public class Platform extends Thread{

    public long initialTime;
    Selector selector;

    Platform () {
        log("Platform: Creating Platform");
        try {
            selector = Selector.open();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void log (String s) {
        Logger logger = new Logger();
        synchronized (logger) {
            Logger.log(s);
        }
    }

    public static void logdebug (String s) {
        Logger logger = new Logger();
        synchronized (logger) {
            // Logger.log(s);
```

```
        }
    }

    public static void externalRegister(Client client){
        logdebug("Platform: Enqueueing registration: "+ client.name);
        Program.registrationQueue.add(client);
    }

    SelectionKey internalRegister(Client client){
        SelectionKey key = null;
        logdebug("Platform: Registering "+ client.name + " " + client.ops);
        try {
            key = client.socketChannel.register(selector, client.ops);
        } catch (ClosedChannelException e) {
            e.printStackTrace();
        }
        key.attach(client);
        return key;
    }

    public static void externalSocketClose(SocketChannel socketChannel){
        logdebug("Platform: Enqueueing close request: " + socketChannel);
        Program.socketCloseQueue.add(socketChannel);
    }

    void internalSocketClose(SocketChannel socketChannel){
        logdebug("Platform: Closing socketChannel: "+ socketChannel);
        try {
            socketChannel.close();
        } catch (IOException e) {
        e.printStackTrace();
        }
    }

    public void run () {
        int step = 0;
        int j = 0;
        int bytesRead = 0;
        int readyChannels;
        while (true) {
//            log("Platform: Step " + step++);
```

```
            try {
                while (!Program.socketCloseQueue.isEmpty()) {
                    SocketChannel socketChannel = Program.socketCloseQueue.remove();
                    internalSocketClose(socketChannel);
                }

                while (!Program.registrationQueue.isEmpty()) {
                    Client client = Program.registrationQueue.remove();
                    internalRegister(client);
                }

                readyChannels = selector.select(1000);

                Set<SelectionKey> keys = selector.selectedKeys();
                Iterator<SelectionKey> i = keys.iterator();

                if (readyChannels > 0 ) {
                    log("Platform: "+ Thread.currentThread().getName() + ", " + " readyChannels = " +
readyChannels);
                    while (i.hasNext()) {
                        SelectionKey key = i.next();
                        Client client = (Client) key.attachment();
                        logdebug("Platform Loop "+ key.toString() + " " + key.readyOps()+ " "+
client.name);
                        logdebug ("Platform: "+ "Acceptable = "+ key.isAcceptable() + " " + "Connectable =
"+ key.isConnectable()+ " " + "Readable = "+ key.isReadable()+ " " + "Writable = "+ key.isWritable());
                        if (client.socketChannel.isOpen()) {
                            logdebug("Platform Open "+ client.name + " " + client.socketChannel);
                        }
                        if (key.isConnectable()) {
                            log ("Platform: " + client.name + " isConnectable");
                            boolean finished;
                            try {
                                finished = client.socketChannel.finishConnect();

                                if (finished) {
                                    log ("Platform: "+ client.name + " finishConnect positive");
                                } else {
                                    log ("Platform: " + client.name + " finishConnect negative");
                                }
                            } catch (IOException e) {
```

```
                            e.printStackTrace();
                        }
                        client.onConnectable();
                    }
                    if (key.isReadable()) {
                        log ("Platform: " + client.name + " isReadable");
                        ByteBuffer buf = ByteBuffer.allocate(1000);
                        try {
                            if (client.socketChannel.isOpen()) {
                                bytesRead = client.socketChannel.read(buf);
                            }
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        log ("Platform: "+ client.name + ": byteRead = " + bytesRead);
                        if (bytesRead > 0) {
                            client.onReadable (buf, bytesRead);
                        }

                    }

                    logdebug("Platform: Removing "+ key.toString() + " " + i.toString());
                    i.remove();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (j >= 100) System.exit(0);
    }
  }
}
```

The Eclipse Project source files – Client.java

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.SelectionKey;
import java.nio.channels.SocketChannel;

public class Client {
    String name;
    SocketChannel socketChannel;
    int ops;

    Client(String name){
        this.name = name;
    }

    void doConnect() throws IOException{
        Platform.log(this.name + " Opening socket");
        socketChannel = SocketChannel.open();
        socketChannel.configureBlocking(false);
        ops = SelectionKey.OP_CONNECT+ SelectionKey.OP_READ;
//        Program.platform.externalRegister(this);
        InetSocketAddress myAddr = new InetSocketAddress("localhost", 8832);
        socketChannel.connect (myAddr);
        Program.platform.externalRegister(this);

    }

    void onConnectable() {
        DoOnConnectable doOnConnectable = new DoOnConnectable(this);
        doOnConnectable.start();
    }

    void onReadable (ByteBuffer buf, int bytesRead) {
        DoOnReadable doOnReadable = new DoOnReadable(buf, bytesRead, this);
        doOnReadable.start();
    }
}
```

The Eclipse Project source files – DoOnConnectable.java

```
public class DoOnConnectable extends Thread{
    Client client;
    DoOnConnectable(Client client){
        this.client=client;
    }
    public void run(){
        Platform.log(client.name + " DoOnConnectable");
    }
}
```

The Eclipse Project source files – DoOnReadable.java

```
import java.nio.ByteBuffer;
public class DoOnReadable extends Thread {
    Client client;
    ByteBuffer buf;
    int bytesRead;
    public DoOnReadable(ByteBuffer buf, int bytesRead, Client client) {
        this.buf = buf;
        this.bytesRead = bytesRead;
        this.client=client;
    }
    public void run(){
        Platform.log(client.name + " DoOnReadble");
        String string = new String (buf.array(), 0, bytesRead);
        Platform.log(client.name + " Received: "+ string);
        if (string.contains("*")) {
            Platform.log(client.name + " exits");
            Platform.externalSocketClose(client.socketChannel);
        }
    }
}
```

The Eclipse Project source files – Logger.java

```
public class Logger {
      public static void log (String s) {
            long current_relative_time = System.currentTimeMillis()- Program.initialTime;
            System.out.println(current_relative_time + ":" + s);
      }
}
```

DynamicClassLoading-Reflection

The DynamicClassLoading-Reflection folder includes the following subfolders:

- ClassLoader-New-vs-FIleSystem
- ClassLoader-Application_Server
- Reflection

ClassLoader-New-vs-FileSystem

This is a simple Java program that loads a class dynamically in two different ways.

- Command line option -c: Regular class loading process;
- Command line option -n: Loading a class from the file system, creation of a new object as is an instance of such a class and casting loaded object on the new object.

The program also shows:

1. The usage of the CLI to compile and execute the Java program
2. The usage of the apache commons-cli library for command line analysis
3. The usage of the CLI for the creation of a fat-jar which includes the dependencies through the unpacking of the apache commons-cli library and the repacking of the application in a single jar file.

How to run the experiment:

1. The clean batch removes all the existing classes and jars and create a directory (./out) which will include the output files. Just type clean.
2. The compile batch compiles the test programs. Just type compile.
3. Execution
    a. First alternative: Run the class directly.
        - Use the run-class batch. Type run-class and give it one of the following options:
          -n to use new() -c to use explicit call loading from file system
    b. Second alternative: Prepare a fat-jar and run the program.
        - Use the prepare-lib batch to copy the apache cli library to the out directory. Just type prepare-lib.
        - Use the build-jar batch to create a fat-jar. Just type build-jar.
        - Use thr run-jar batch. Type run-jar and give it one of the following options:
          -n to use new() -c to use explicit call loading from file system

The following pages show the source code of the programs, A.java and B.java. The manifest files includes the following content:

MANIFEST.FM

```
Manifest-Version: 1.0
Class-Path: .
Main-Class: A
```

```
import org.apache.commons.cli.*;
@SuppressWarnings( "deprecation" )
public class A {
    public static void main(String[] args) throws Exception {

        System.out.println("A: Starting... ");
        A a = new A();

        Options options = new Options();
        options.addOption("n", "--new", false, "create a dynamic object using new()");
        options.addOption("c", "--create", false, "create a dynamic explicitly");

        CommandLineParser parser = new DefaultParser();

        try {
            CommandLine cmd = parser.parse(options, args);
        if(cmd.hasOption("n")) {
            System.out.println("Creating the object using new()...");
            a.createObject();
        }
        if(cmd.hasOption("c")) {
            System.out.println("Loading the object explicitly from the File System ...");
            a.getObjectFromFileSystem();
        }
        } catch (ParseException e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("A: Done... ");
    }

    public void createObject() {
        B b1 = null;
        b1 = new B();
        b1.doSomething();
    }

    public void getObjectFromFileSystem() {
        B b2 = null;
```

```
        ClassLoader cl = A.class.getClassLoader();
        try {
            Class<?> BClass = cl.loadClass("B");
            Object o = BClass.newInstance();
            b2 = (B) o;
//            b2 = (B) BClass.newInstance();
        }
        catch (Exception e) {
            System.out.println("Error");
        }
        b2.doSomething();
    }
}
```

B.java

```
public class B {
    public void doSomething() {
        System.out.println("B: Doing its job ...");
    }
}
```

ClassLoader-Application_Server

This directory includes two subdirectories:
The first subdirectory, called AppServDir, is the one in which the Application Server works. It includes the Application Server source files, or more specifically that part of the Application Server that loads the external class, inspects it to confirm that it is an implementations of the appropriate interface and , if it is, calls some of its methods to show that it works.
The source files are the following:

- AppServer.java, the main program.
- MyClassLoader.java, that takes care of Class Loading
- ClassToBeLoadedInterface.java, the reference interface, which is shared with the other directory, where the classes to be loaded reside.

The second subdirectory, called DeveloperDir, is the one in which the developer creates the class to be loaded. The source files are the following:

- ClassToBeLoaded.java, the class to be loaded, implementing the appropriate interface.
- ClassToBeLoadedInterface.java, the reference interface, which is shared with the other directory, where the Application Server resides.

The example works as follows:
You have to compile the two programs in the two subdirectories separately.

1. In AppServerDir you compile the Application Server, using the following command:
   javac ClassLoaderExample.java (a batch file is also provided)
2. In DeveloperDir you compile the class to be loaded using the following command:
   Javac ClassToBeLoaded.java (a batch file is also provided)

Then you can copy the class generated (ClassToBeLoaded.class) to the AppServerDir subdirectory using the following command:
copy …. DeveloperDir\ClassToBeLoaded.class ….\AppServ\ClassToBeLoaded.class (a batch file is also provided)
Now you can execute the Dynamic Loading of the class using the following command:
Java ClassLoaderExample (oppure con il batch)

The proposed test can be carried out as follows:

1. You compile the class to be loaded maintaining “implements ClassToBeLoadedInterface” in the definition of the ClassToBeLoaded class and look at how the Application Server works.
2. The you comment the clause “implements ClassToBeLoadedInterface” to exclude the statement and look at how the Application Server works.

AppServDir: AppServer.java

```
/*
 This file uses the instanceof statement to check that the class to be loaded
implements the apporpriate interface
*/
import java.lang.Object;
import java.io.File;
import javax.management.*;
@SuppressWarnings( "deprecation" )
public class AppServer {
      public static void main (String args []) {
        boolean loaded = false;
        Class myClass = null;
        String fileList[];
        File newFile = new File (".");
        fileList=newFile.list();

        MyClassLoader cL = new MyClassLoader();

        for (int i=0; i < fileList.length; i++) {
            if (fileList[i].equals("ClassToBeLoaded.class")){
                System.out.println("Loading "+ fileList[i]+"...");
                loaded=true;
                try {
                    myClass = cL.loadClass("ClassToBeLoaded");
                } catch (Exception e) {
                    System.out.println(e.toString());
                    System.exit(1);
                }
            }
        }
        if (loaded == false) {
            System.out.println("ClassToBeLoaded not found. Exiting");
        } else {
            try {

                Object object = myClass.newInstance();
                System.out.println("object istanceof ClassToBeLoadedInterface:" + (object instanceof
ClassToBeLoadedInterface));
                if (object instanceof ClassToBeLoadedInterface){
                    ClassToBeLoadedInterface myObject = (ClassToBeLoadedInterface) object;
```

```
                    int i =35;
                    myObject.set_value (i);
                    System.out.println("Access to myObject: set_value("+i+")");
                    int j = myObject.get_value();
                    System.out.println("Access to myObject: get_value()="+ j);
                } else {
                    System.out.println ("Error: Class Loaded is not an instance of
ClassToBeLoadedInterface");
                }
            } catch (Exception e){
                System.out.println(e.toString());
            }
        }
    }
}
```

AppServDir: ClassToBeLoadedInterface.java

```
interface ClassToBeLoadedInterface {
    void set_value (int value);
    int get_value();
    public void print_value(String param);
}
```

AppServDir: MyClassLoader.java

```
import javax.management.loading.ClassLoaderRepository;
public class MyClassLoader extends ClassLoader {
}
```

DeveloperDir: ClassToBeLoaded.java

```
public class ClassToBeLoaded implements ClassToBeLoadedInterface{
    private int value = 0;

    public void set_value(int value) {
        this.value=value;
    }

    public int get_value (){
        return this.value;
    }

    public void print_value(String string) {
        System.out.println (string+" "+ this.value);
    }
}
```

DeveloperDir: ClassToBeLoadedInterface.java

```
interface ClassToBeLoadedInterface {
    void set_value (int value);
    int get_value();
    public void print_value(String param);
}
```

Reflection

Reflection refers to the ability of a program to analyze its code and/or the other programs. More specifically it allows programs to analyze code elements which are available at run time, i.e., written “after” the program that performs the analysis.

Reflection is a core element of Software Platforms, as Software Platforms typically require the integration of software modules implemented by third parties after platform deployment.

The main classes are: Class, Method, Constructor, Field, Type, Annotation

The following program (`DumpEverything.java`) runs reflection on a program, for example on the Test class (`Test.java`)

```
import java.lang.reflect.*;
import java.lang.annotation.Annotation;

// This is a program to test Java ReflectionException

interface MyAnnotation {
   String name();
   String value();
}
   public class DumpEverything {
      public static void main(String args[])
      {
          System.out.println(args[0]);
        try {

            // Class Loading
            Class c = Class.forName(args[0]);

            // Get Declared Methods
            Method methodlist[] = c.getDeclaredMethods();
            // For each method print name, types of parameters, types of exceptions
            for (int i = 0; i < methodlist.length; i++) {
               Method method = methodlist[i];
               System.out.println("Method Name = " + method.getName());
               Class parameterTypes[] = method.getParameterTypes();
               for (int j = 0; j < parameterTypes.length; j++)
                  System.out.println("Type of Parameter #" + j + " " + parameterTypes[j]);
               Class exceptionTypes[] = method.getExceptionTypes();
               for (int j = 0; j < exceptionTypes.length; j++)
                  System.out.println("Type of Exception #" + j + " " + exceptionTypes[j]);
               System.out.println("return type = " + method.getReturnType());
               System.out.println("-----");
            }

            // Get Contructors
            Constructor constructorList[] = c.getDeclaredConstructors();
            // For each constructuor print name, types of parameters, types of exceptions
            for (int i = 0; i < constructorList.length; i++) {
               Constructor constructor = constructorList[i];
               System.out.println("Constructor Name = " + constructor.getName());
```

```
            Class parameterTypes[] = constructor.getParameterTypes();
            for (int j = 0; j < parameterTypes.length; j++)
               System.out.println("Type of Parameter #" + j + " " + parameterTypes[j]);
            Class exceptionTypes[] = constructor.getExceptionTypes();
            for (int j = 0; j < exceptionTypes.length; j++)
               System.out.println("Type of Exception #" + j + " " + exceptionTypes[j]);
            System.out.println("-----");
         }

         // Get Fields
         Field fieldList[] = c.getDeclaredFields();
         // For each field print name, type, modifiers
         for (int i = 0; i < fieldList.length; i++) {
            Field field = fieldList[i];
            System.out.println("Field Name = " + field.getName());
            System.out.println("Field Type = " + field.getType());
            int fieldModifiers = field.getModifiers();
            System.out.println("Field Modifiers = " + Modifier.toString(fieldModifiers));
            System.out.println("-----");
         }

         // Get Annotations
         Annotation annotationList[] = c.getAnnotations();
         // For each field print name, type, modifiers
         for (int i = 0; i < annotationList.length; i++) {
            Annotation annotation = annotationList[i];
            System.out.println("Annotation = " + annotation.toString());

            MyAnnotation a = (MyAnnotation) annotation;
            System.out.println(a.name() + " " + a.value());
            System.out.println("-----");
         }

         } catch (Throwable e) {
         System.err.println(e);
      }
   }
}
```

Test.java

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotation {
   String name();
   String value();
}

@MyAnnotation (name = "MyName", value = "MyValue")
public class Test{
    public String mymethod(int i){
        return "Called with int parameter";
    }
    public String mymethod (float f){
        return "Called with float parameter";
    }
}
```

Servlet examples

Here is a description of how to use the two programs. Both programs require that directory C:\Temp (it can be reconfigured of course) includes the following files:

- static.html, used by both programs as a static resource. It can be loaded by issuing a request to http://localhost:8080/static.html
- HelloWorld.class, used by MyHttpServer0 as a servlet. It can activated by issuing a request to http://localhost:8080/servlet/HelloWorld
- Myservlet directory, used by MyHttpServer1 as a servlet repository. It contains a directory named myservlet, which includes a metadata file and a the HelloWorld.class servlet.

The directory named referencedirectory in the Aulaweb repository can be taken as a reference.

MyHttpServer0, the first of the two programs, aims at showing the distinction between a static web access and a dynamic web access (associated to a /servlet keyword in the URL). When receiving a request, the server checks whether it is a static request or a dynamic request and activates the appropriate processor. In case of dynamic requests the program loads the required class (e.g., HelloWorld if http://localhost:8080/servlet/HelloWorld was indicated as a target URL) and activates it.

MyHttpServer1 is an evolution toward a platform, i.e. an automated servlet container. In particular it distinguishes class management from class activation. A Management Console, based on CLI, allows loading/unloading servlets, where each servlet must correspond to a different directory in C:/Temp/servletrepository (configurable). For example, the myservlet directory includes two files, the first named metadata.txt and the second named HelloWorld.class. The metadata.txt file associates the name of class to a servlet.

You can use the two programs as follows:

1. Place the static.html file in the C:\Temp directory and the HelloWorld.class and the myservlet directory in the C:\Temp\servletrepository directory. If you use another directory as a root (C:\Temp) you have to reconfigure the programs.

Then run the first program (MyHttpServer0). The program does not require commands. Just run it. Then run a browser and access the following URLS:

http://localhost:8080/static.html to retrieve the static content

http://localhost:8080/servlet/HelloWorld to activate the servlet

Then run the second program. You should see a CLI based management console. If you do not type any command you should be able to access the static content in the same way as in the previous case, while you should not allowed to activate any servlet. Now load a servlet through the following command

load myservlet

Then you should be able to activate the servlet by typing the following command:

http://localhost:8080/servlet/myservlet

Remember that when you activate servlet myservlet, you are activating class HelloWorld.

HelloWorld.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class HelloWorld extends HttpServlet {

   private String message;
   public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
            message = "HTTP/1.1 200 OK\r\n"+
                "Content-Type: text/html\r\n"+
                "Content-Length: 22\r\n" +
                "\r\n"+
                   "<h1>Here I am!!!!</h1>";
      PrintWriter out = response.getWriter();
      out.println(message);

   }
}
```

MyHttpServer0: MyHttpServer.java

```
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class MyHttpServer {

    public static final String WEB_ROOT = "C:/Temp";
    private static final String SHUTDOWN_COMMAND = "/SHUTDOWN";
    private boolean shutdown = false;
    public static void  main(String[] args) {
        MyHttpServer server = new MyHttpServer();
        System.out.println(WEB_ROOT);
        server.await();
    }
    public void await() {
        ServerSocket serverSocket = null;
        int port = 8080;
        try {
            serverSocket = new ServerSocket(port, 1, InetAddress.getByName("127.0.0.1"));
        }
        catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
        while (!shutdown) {
            Socket socket = null;
            InputStream input = null;
            OutputStream output = null;
            try {
                socket = serverSocket.accept();
                input = socket.getInputStream();
                output = socket.getOutputStream();
                Request request = new Request(input);
                request.parse();
                Response response = new Response(output);
                response.setRequest(request);
```

```
                if (request.getUri() != null) {
                    if (request.getUri().startsWith("/servlet")) {
                        MyServletProcessor processor = new MyServletProcessor();
                        processor.process(request, response);
                    }
                    else {
                        MyStaticResourceProcessor processor = new MyStaticResourceProcessor();
                        processor.process(request, response);
                    }
                    socket.close();
                    shutdown = request.getUri().equals(SHUTDOWN_COMMAND);
                    }
                }
                catch (Exception e) {
                    e.printStackTrace();
                    continue;
                }
            }
        }
}
```

MyHttpServer0: MyServletProcessor.java

```
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.net.URLClassLoader;
import java.net.URLStreamHandler;
import javax.servlet.Servlet;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServletProcessor {
    public void process(Request request, Response response) {

        String repository = new String("file:" + MyHttpServer.WEB_ROOT +
File.separator+"servletrepository"+ File.separatorChar);
        String uri = request.getUri();
        String servletName = uri.substring(uri.lastIndexOf("/") + 1);

        URLClassLoader loader = null;
        try {
            URL[] urls = new URL[1];
            urls[0] = new URL(repository);
            loader = new URLClassLoader(urls);
        }
        catch (IOException e) {
            System.out.println(e.toString() );
        }

        Class myClass = null;
        try {
            myClass = loader.loadClass(servletName);
        }
        catch (ClassNotFoundException e) {
            System.out.println("Class "+ servletName + " not found");
            return;
        }
```

```
        HttpServlet servlet = null;
        try {
            servlet = (HttpServlet) myClass.newInstance();
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
        try {
            servlet.service((ServletRequest) request, (ServletResponse) response);
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
        catch (Throwable e) {
            System.out.println(e.toString());
        }
    }
}
```

MyHttpServer0: MyStaticResourceProcessor.java

```
import java.io.IOException;
public class MyStaticResourceProcessor {
    public void process(Request request, Response response) {
        try {
            response.sendStaticResource();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MyHttpServer0: Request.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.security.Principal;
import java.util.Collection;
import java.util.Enumeration;
import java.util.Locale;
import java.util.Map;

import javax.servlet.AsyncContext;
import javax.servlet.DispatcherType;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpUpgradeHandler;
import javax.servlet.http.Part;

public class Request implements HttpServletRequest {
    private InputStream input;
    private String uri;
    public Request(InputStream input) {
        this.input = input;
    }
    public String getUri() {
        return uri;
    }
    private String parseUri(String requestString) {
        int index1, index2;
        index1 = requestString.indexOf(' ');
        if (index1 != -1) {
            index2 = requestString.indexOf(' ', index1 + 1);
            if (index2 > index1)
```

```
            return requestString.substring(index1 + 1, index2);
    }
    return null;
}
public void parse() {
    StringBuffer request = new StringBuffer(2048);
    int i;
    byte[] buffer = new byte[2048];
    try {
        i = input.read(buffer);
    }
    catch (IOException e) {
        e.printStackTrace();
        i = -1;
    }
    for (int j=0; j<i; j++) {
        request.append((char) buffer[j]);
    }
//    System.out.print(request.toString());
    uri = parseUri(request.toString());
}

public Object getAttribute(String attribute) {
    return null;
}
public Enumeration getAttributeNames() {
    return null;
}
public String getRealPath(String path) {
    return null;
}
public RequestDispatcher getRequestDispatcher(String path) {
    return null;
}
public boolean isSecure() {
    return false;
}
public String getCharacterEncoding() {
    return null;
}
public int getContentLength() {
```

```
    return 0;
}
public String getContentType(){
    return null;
}
public ServletInputStream getInputStream() throws IOException {
    return null;
}
public Locale getLocale() {
    return null;
}
public Enumeration getLocales() {
    return null;
}
public String getParameter(String name) {
    return null;
}
public Map getParameterMap() {
    return null;
}
public Enumeration getParameterNames() {
    return null;
}
public String[] getParameterValues(String parameter) {
    return null;
}
public String getProtocol() {
    return null;
}
public BufferedReader getReader() throws IOException {
    return null;
}
public String getRemoteAddr() {
    return null;
}
public String getRemoteHost() {
    return null;
}
public String getScheme() {
    return null;
}
```

```
public String getServerName() {
    return null;
}
public int getServerPort() {
    return 0;
}
public void removeAttribute(String attribute) { }
public void setAttribute(String key, Object value) { }
public void setCharacterEncoding(String encoding)
throws UnsupportedEncodingException { }
@Override
public long getContentLengthLong() {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public int getRemotePort() {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public String getLocalName() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getLocalAddr() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public int getLocalPort() {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public ServletContext getServletContext() {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public AsyncContext startAsync() throws IllegalStateException {
    // TODO Auto-generated method stub
    return null;
}
@Override
public AsyncContext startAsync(ServletRequest servletRequest, ServletResponse servletResponse)
        throws IllegalStateException {
    // TODO Auto-generated method stub
    return null;
}
@Override
public boolean isAsyncStarted() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isAsyncSupported() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public AsyncContext getAsyncContext() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public DispatcherType getDispatcherType() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getAuthType() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public Cookie[] getCookies() {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public long getDateHeader(String name) {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public String getHeader(String name) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public Enumeration<String> getHeaders(String name) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public Enumeration<String> getHeaderNames() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public int getIntHeader(String name) {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public String getMethod() {
    // TODO Auto-generated method stub
    String s = "GET";
    return s;
}
@Override
public String getPathInfo() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getPathTranslated() {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public String getContextPath() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getQueryString() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getRemoteUser() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public boolean isUserInRole(String role) {
    // TODO Auto-generated method stub
    return false;
}
@Override
public Principal getUserPrincipal() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getRequestedSessionId() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getRequestURI() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public StringBuffer getRequestURL() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getServletPath() {
```

```
    // TODO Auto-generated method stub
    return null;
}
@Override
public HttpSession getSession(boolean create) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public HttpSession getSession() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String changeSessionId() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public boolean isRequestedSessionIdValid() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isRequestedSessionIdFromCookie() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isRequestedSessionIdFromURL() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isRequestedSessionIdFromUrl() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean authenticate(HttpServletResponse response) throws IOException, ServletException {
    // TODO Auto-generated method stub
```

```
        return false;
    }
    @Override
    public void login(String username, String password) throws ServletException {
        // TODO Auto-generated method stub

    }
    @Override
    public void logout() throws ServletException {
        // TODO Auto-generated method stub

    }
    @Override
    public Collection<Part> getParts() throws IOException, ServletException {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Part getPart(String name) throws IOException, ServletException {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public <T extends HttpUpgradeHandler> T upgrade(Class<T> httpUpgradeHandlerClass)
            throws IOException, ServletException {
        // TODO Auto-generated method stub
        return null;
    }
}
```

MyHttpServer0: Response.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Collection;
import java.util.Locale;
import javax.servlet.ServletOutputStream;
import javax.servlet.ServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;

public class Response implements HttpServletResponse {
    private static final int BUFFER_SIZE = 1024;
    Request request;
    OutputStream output;
    PrintWriter writer;
    public Response(OutputStream output) {
        this.output = output;
    }
    public void setRequest(Request request) {
        this.request = request;
    }
    /* This method is used to serve static pages */
    public void sendStaticResource() throws IOException {
        byte[] bytes = new byte[BUFFER_SIZE];
        FileInputStream fis = null;
        try {
            /* request.getUri has been replaced by request.getRequestURI */
            File file = new File(MyHttpServer.WEB_ROOT, request.getUri());
            fis = new FileInputStream(file);

            int ch = fis.read(bytes, 0, BUFFER_SIZE);
            while (ch!=-1) {
                output.write(bytes, 0, ch);
                ch = fis.read(bytes, 0, BUFFER_SIZE);
            }
        }
```

```
    catch (FileNotFoundException e) {
        String errorMessage = "HTTP/1.1 404 File Not Found\r\n" +
        "Content-Type: text/html\r\n" +
        "Content-Length: 23\r\n" +
        "\r\n" +
        "<h1>File Not Found</h1>";
        output.write(errorMessage.getBytes());
    }
    finally {
        if (fis!=null)
            fis.close();
    }
}
/** implementation of ServletResponse */
public void flushBuffer() throws IOException { }
public int getBufferSize() {
    return 0;
}
public String getCharacterEncoding() {
    return null;
}
public Locale getLocale() {
    return null;
}
public ServletOutputStream getOutputStream() throws IOException {
    return null;
}
public PrintWriter getWriter() throws IOException {
    // autoflush is true, println() will flush,
    // but print() will not.
    writer = new PrintWriter(output, true);
    return writer;
}
public boolean isCommitted() {
    return false;
}
public void reset() { }
public void resetBuffer() { }
public void setBufferSize(int size) { }
public void setContentLength(int length) { }
public void setContentType(String type) { }
```

```
public void setLocale(Locale locale) { }
@Override
public String getContentType() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public void setCharacterEncoding(String arg0) {
    // TODO Auto-generated method stub

}
@Override
public void setContentLengthLong(long arg0) {
    // TODO Auto-generated method stub

}
@Override
public void addCookie(Cookie cookie) {
    // TODO Auto-generated method stub

}
@Override
public boolean containsHeader(String name) {
    // TODO Auto-generated method stub
    return false;
}
@Override
public String encodeURL(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String encodeRedirectURL(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String encodeUrl(String url) {
    // TODO Auto-generated method stub
    return null;
}
```

```
@Override
public String encodeRedirectUrl(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public void sendError(int sc, String msg) throws IOException {
    // TODO Auto-generated method stub

}
@Override
public void sendError(int sc) throws IOException {
    // TODO Auto-generated method stub

}
@Override
public void sendRedirect(String location) throws IOException {
    // TODO Auto-generated method stub

}
@Override
public void setDateHeader(String name, long date) {
    // TODO Auto-generated method stub

}
@Override
public void addDateHeader(String name, long date) {
    // TODO Auto-generated method stub

}
@Override
public void setHeader(String name, String value) {
    // TODO Auto-generated method stub

}
@Override
public void addHeader(String name, String value) {
    // TODO Auto-generated method stub

}
@Override
```

```
    public void setIntHeader(String name, int value) {
        // TODO Auto-generated method stub

    }
    @Override
    public void addIntHeader(String name, int value) {
        // TODO Auto-generated method stub

    }
    @Override
    public void setStatus(int sc) {
        // TODO Auto-generated method stub

    }
    @Override
    public void setStatus(int sc, String sm) {
        // TODO Auto-generated method stub

    }
    @Override
    public int getStatus() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public String getHeader(String name) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Collection<String> getHeaders(String name) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Collection<String> getHeaderNames() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

MyHttpServer1: ManagementConsole.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.Hashtable;

import javax.servlet.http.HttpServlet;

public class ManagementConsole extends Thread {
    ManagementConsole(){
    }

    String firstWord(String command) {
        if (command.contains(" ")){
            int index = command.indexOf(" ");
            return command.substring(0, index);
        } else return command;

    }

    String secondWord(String command) {
        if (command.contains(" ")) {
            int index = command.indexOf(" ");
            return command.substring(index+1, command.length());
            } else {
                return command;
            }
    }

    void executeUnload(String servletInternalName){
        if (!ServletHashTable.contains(servletInternalName)) {
            System.out.println("Servlet " + servletInternalName + " not in the servlet repository");
        } else {
            ServletHashTable.remove(servletInternalName);
            System.out.println("Servlet " + servletInternalName + " removed");
        }
```

```
}
void executeLoad(String servletInternalName){
    if (ServletHashTable.contains(servletInternalName)) {
        System.out.println("Servlet " + servletInternalName + " already in the servlet repository");
    } else {
        String servletClassName = null;
        String servletRepository = new String("C:/Temp/servletrepository" + File.separator);
        String servletDir = new String(servletRepository + File.separator+ servletInternalName);
        File f = new File(servletDir );
        if (!(f.exists() && f.isDirectory())) {
            System.out.println("Directory " + servletDir + " does not exists");
            return;
        } else {
            FileReader fr=null;
            try
            {
                String metadataFile = servletDir+File.separator+"metadata.txt";
                BufferedReader reader = new BufferedReader(new FileReader(metadataFile));
                String command = reader.readLine();
                while (command != null) {
                    if (command.contains("=")){
                        int index = command.indexOf("=");
                        servletClassName = command.substring(index+1, command.length());
                    }
                    command = reader.readLine();
                }
            }
            catch (FileNotFoundException fe)
            {
                System.out.println("File not found");
            } catch (IOException e) {
                e.printStackTrace();
            }

            URLClassLoader loader = null;
            try {
                URL[] urls = new URL[1];
                urls[0] = new URL("file:" + servletDir + File.separator);
                loader = new URLClassLoader(urls);
            }
            catch (IOException e) {
```

```
                System.out.println(e.toString() );
            }

            Class myClass = null;
            try {
                myClass = loader.loadClass(servletClassName);
            }
            catch (ClassNotFoundException e) {
                System.out.println(e.toString());
            }
            HttpServlet servlet = null;
            try {
                servlet = (HttpServlet) myClass.newInstance();
            }
            catch (Exception e) {
                System.out.println(e.toString());
            }
            ServletHashTable.put(servletInternalName, servlet);
            System.out.println("Servlet " + servletInternalName + " added");
        }
    }
}

void executeCommand(String command) {
    if (firstWord(command).equals("load")) {
        executeLoad(secondWord(command));
        return;
    }
    if (firstWord(command).equals("unload")) {
        executeUnload(secondWord(command));
        return;
    }

}
public void run() {
    String command= null;
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
    System.out.print ("Command: ");
    try {
        command  = bufferedReader.readLine();
    } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        executeCommand(command);
        while (!command.equals("quit")){
            System.out.print ("Command: ");
            try {
                command  = bufferedReader.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            executeCommand(command);
        }
        Shutdown.flag=true;
    }
}
```

MyHttpServer1: MyHttpServer.java

```
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Hashtable;

import javax.servlet.http.HttpServlet;

public class MyHttpServer {
    public static final String WEB_ROOT = "    C:/Temp";

    private static final String SHUTDOWN_COMMAND = "/SHUTDOWN";
    private boolean shutdown = false;
    public static void  main(String[] args) {
        ServletHashTable servletHashTable = new ServletHashTable();

        ManagementConsole managementConsole = new ManagementConsole();
        managementConsole.start();

        MyHttpServer myHttpServer = new MyHttpServer();
        myHttpServer.await();
        System.out.println("Exiting...");
    }

    MyHttpServer() {
    }

    public void await() {
        ServerSocket serverSocket = null;
        int port = 8080;
        try {
            serverSocket = new ServerSocket(port, 1, InetAddress.getByName("127.0.0.1"));
        }
        catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
```

```
        while (!shutdown ) {
            Socket socket = null;
            InputStream input = null;
            OutputStream output = null;
            try {
                socket = serverSocket.accept();
                if (Shutdown.flag) {
                    return;
                }
                input = socket.getInputStream();
                output = socket.getOutputStream();
                Request request = new Request(input);
                request.parse();
                Response response = new Response(output);
                response.setRequest(request);
                if (request.getUri() != null) {
                    if (request.getUri().startsWith("/servlet")) {
                        MyServletProcessor processor = new MyServletProcessor();
                        processor.process(request, response);
                    }
                    else {
                        MyStaticResourceProcessor processor = new MyStaticResourceProcessor();
                        processor.process(request, response);
                    }
                    shutdown = request.getUri().equals(SHUTDOWN_COMMAND);
                    socket.close();
                }
            }
            catch (Exception e) {
                e.printStackTrace();
                continue;
            }
        }
    }
}
```

MyHttpServer1: MyServletProcessor.java

```
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.net.URLClassLoader;
import java.net.URLStreamHandler;
import javax.servlet.Servlet;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServletProcessor {
    public void process(Request request, Response response) {
        String uri = request.getUri();
        String servletName = uri.substring(uri.lastIndexOf("/") + 1);
        if (!ServletHashTable.contains(servletName)) {
    //        System.out.println("Error: " + servletName + " unknown");
        } else {
            HttpServlet servlet = ServletHashTable.get(servletName);
            try {
                servlet.service((ServletRequest) request, (ServletResponse) response);
            }
            catch (Exception e) {
                System.out.println(e.toString());
            }
            catch (Throwable e) {
                System.out.println(e.toString());
            }
        }
    }
}
```

MyHttpServer1: MyStaticResourceProcessor.java

```
import java.io.IOException;
public class MyStaticResourceProcessor {
    public void process(Request request, Response response) {
        try {
            response.sendStaticResource();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

MyHttpServer1: Request.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.UnsupportedEncodingException;
import java.security.Principal;
import java.util.Collection;
import java.util.Enumeration;
import java.util.Locale;
import java.util.Map;

import javax.servlet.AsyncContext;
import javax.servlet.DispatcherType;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.http.HttpUpgradeHandler;
import javax.servlet.http.Part;

public class Request implements HttpServletRequest {
    private InputStream input;
    private String uri;
    public Request(InputStream input) {
        this.input = input;
    }
    public String getUri() {
        return uri;
    }
    private String parseUri(String requestString) {
        int index1, index2;
        index1 = requestString.indexOf(' ');
        if (index1 != -1) {
            index2 = requestString.indexOf(' ', index1 + 1);
            if (index2 > index1)
```

```
            return requestString.substring(index1 + 1, index2);
    }
    return null;
}
public void parse() {
    StringBuffer request = new StringBuffer(2048);
    int i;
    byte[] buffer = new byte[2048];
    try {
        i = input.read(buffer);
    }
    catch (IOException e) {
        e.printStackTrace();
        i = -1;
    }
    for (int j=0; j<i; j++) {
        request.append((char) buffer[j]);
    }
//    System.out.print(request.toString());
    uri = parseUri(request.toString());
}

public Object getAttribute(String attribute) {
    return null;
}
public Enumeration getAttributeNames() {
    return null;
}
public String getRealPath(String path) {
    return null;
}
public RequestDispatcher getRequestDispatcher(String path) {
    return null;
}
public boolean isSecure() {
    return false;
}
public String getCharacterEncoding() {
    return null;
}
public int getContentLength() {
```

```
    return 0;
}
public String getContentType(){
    return null;
}
public ServletInputStream getInputStream() throws IOException {
    return null;
}
public Locale getLocale() {
    return null;
}
public Enumeration getLocales() {
    return null;
}
public String getParameter(String name) {
    return null;
}
public Map getParameterMap() {
    return null;
}
public Enumeration getParameterNames() {
    return null;
}
public String[] getParameterValues(String parameter) {
    return null;
}
public String getProtocol() {
    return null;
}
public BufferedReader getReader() throws IOException {
    return null;
}
public String getRemoteAddr() {
    return null;
}
public String getRemoteHost() {
    return null;
}
public String getScheme() {
    return null;
}
```

```
public String getServerName() {
    return null;
}
public int getServerPort() {
    return 0;
}
public void removeAttribute(String attribute) { }
public void setAttribute(String key, Object value) { }
public void setCharacterEncoding(String encoding)
throws UnsupportedEncodingException { }
@Override
public long getContentLengthLong() {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public int getRemotePort() {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public String getLocalName() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getLocalAddr() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public int getLocalPort() {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public ServletContext getServletContext() {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public AsyncContext startAsync() throws IllegalStateException {
    // TODO Auto-generated method stub
    return null;
}
@Override
public AsyncContext startAsync(ServletRequest servletRequest, ServletResponse servletResponse)
        throws IllegalStateException {
    // TODO Auto-generated method stub
    return null;
}
@Override
public boolean isAsyncStarted() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isAsyncSupported() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public AsyncContext getAsyncContext() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public DispatcherType getDispatcherType() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getAuthType() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public Cookie[] getCookies() {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public long getDateHeader(String name) {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public String getHeader(String name) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public Enumeration<String> getHeaders(String name) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public Enumeration<String> getHeaderNames() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public int getIntHeader(String name) {
    // TODO Auto-generated method stub
    return 0;
}
@Override
public String getMethod() {
    // TODO Auto-generated method stub
    String s = "GET";
    return s;
}
@Override
public String getPathInfo() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getPathTranslated() {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public String getContextPath() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getQueryString() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getRemoteUser() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public boolean isUserInRole(String role) {
    // TODO Auto-generated method stub
    return false;
}
@Override
public Principal getUserPrincipal() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getRequestedSessionId() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getRequestURI() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public StringBuffer getRequestURL() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String getServletPath() {
```

```
    // TODO Auto-generated method stub
    return null;
}
@Override
public HttpSession getSession(boolean create) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public HttpSession getSession() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String changeSessionId() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public boolean isRequestedSessionIdValid() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isRequestedSessionIdFromCookie() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isRequestedSessionIdFromURL() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean isRequestedSessionIdFromUrl() {
    // TODO Auto-generated method stub
    return false;
}
@Override
public boolean authenticate(HttpServletResponse response) throws IOException, ServletException {
    // TODO Auto-generated method stub
```

```
        return false;
    }
    @Override
    public void login(String username, String password) throws ServletException {
        // TODO Auto-generated method stub

    }
    @Override
    public void logout() throws ServletException {
        // TODO Auto-generated method stub

    }
    @Override
    public Collection<Part> getParts() throws IOException, ServletException {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Part getPart(String name) throws IOException, ServletException {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public <T extends HttpUpgradeHandler> T upgrade(Class<T> httpUpgradeHandlerClass)
            throws IOException, ServletException {
        // TODO Auto-generated method stub
        return null;
    }
}
```

MyHttpServer1: Response.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Collection;
import java.util.Locale;
import javax.servlet.ServletOutputStream;
import javax.servlet.ServletResponse;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletResponse;

public class Response implements HttpServletResponse {
    private static final int BUFFER_SIZE = 1024;
    Request request;
    OutputStream output;
    PrintWriter writer;
    public Response(OutputStream output) {
        this.output = output;
    }
    public void setRequest(Request request) {
        this.request = request;
    }
    /* This method is used to serve static pages */
    public void sendStaticResource() throws IOException {
        byte[] bytes = new byte[BUFFER_SIZE];
        FileInputStream fis = null;
        try {
            File file = new File("C:/Temp/", request.getUri());
        fis = new FileInputStream(file);

            int ch = fis.read(bytes, 0, BUFFER_SIZE);
            while (ch!=-1) {
                output.write(bytes, 0, ch);
                ch = fis.read(bytes, 0, BUFFER_SIZE);
            }
        }
        catch (FileNotFoundException e) {
```

```
            String errorMessage = "HTTP/1.1 404 File Not Found\r\n" +
            "Content-Type: text/html\r\n" +
            "Content-Length: 23\r\n" +
            "\r\n" +
            "<h1>File Not Found</h1>";
            output.write(errorMessage.getBytes());
        }
        finally {
            if (fis!=null)
                fis.close();
        }
    }
    /** implementation of ServletResponse */
    public void flushBuffer() throws IOException { }
    public int getBufferSize() {
        return 0;
    }
    public String getCharacterEncoding() {
        return null;
    }
    public Locale getLocale() {
        return null;
    }
    public ServletOutputStream getOutputStream() throws IOException {
        return null;
    }
    public PrintWriter getWriter() throws IOException {
        // autoflush is true, println() will flush,
        // but print() will not.
        writer = new PrintWriter(output, true);
        return writer;
    }
    public boolean isCommitted() {
        return false;
    }
    public void reset() { }
    public void resetBuffer() { }
    public void setBufferSize(int size) { }
    public void setContentLength(int length) { }
    public void setContentType(String type) { }
    public void setLocale(Locale locale) { }
```

```
@Override
public String getContentType() {
    // TODO Auto-generated method stub
    return null;
}
@Override
public void setCharacterEncoding(String arg0) {
    // TODO Auto-generated method stub

}
@Override
public void setContentLengthLong(long arg0) {
    // TODO Auto-generated method stub

}
@Override
public void addCookie(Cookie cookie) {
    // TODO Auto-generated method stub

}
@Override
public boolean containsHeader(String name) {
    // TODO Auto-generated method stub
    return false;
}
@Override
public String encodeURL(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String encodeRedirectURL(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public String encodeUrl(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
```

```
public String encodeRedirectUrl(String url) {
    // TODO Auto-generated method stub
    return null;
}
@Override
public void sendError(int sc, String msg) throws IOException {
    // TODO Auto-generated method stub

}
@Override
public void sendError(int sc) throws IOException {
    // TODO Auto-generated method stub

}
@Override
public void sendRedirect(String location) throws IOException {
    // TODO Auto-generated method stub

}
@Override
public void setDateHeader(String name, long date) {
    // TODO Auto-generated method stub

}
@Override
public void addDateHeader(String name, long date) {
    // TODO Auto-generated method stub

}
@Override
public void setHeader(String name, String value) {
    // TODO Auto-generated method stub

}
@Override
public void addHeader(String name, String value) {
    // TODO Auto-generated method stub

}
@Override
public void setIntHeader(String name, int value) {
```

```
        // TODO Auto-generated method stub

    }
    @Override
    public void addIntHeader(String name, int value) {
        // TODO Auto-generated method stub

    }
    @Override
    public void setStatus(int sc) {
        // TODO Auto-generated method stub

    }
    @Override
    public void setStatus(int sc, String sm) {
        // TODO Auto-generated method stub

    }
    @Override
    public int getStatus() {
        // TODO Auto-generated method stub
        return 0;
    }
    @Override
    public String getHeader(String name) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Collection<String> getHeaders(String name) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public Collection<String> getHeaderNames() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

MyHttpServer1: ServletHashTable

```
import java.util.Hashtable;

import javax.servlet.http.HttpServlet;

public class ServletHashTable {
    static Hashtable<String, HttpServlet> ht;
    ServletHashTable() {
        ht = new Hashtable<String, HttpServlet>();
    }
    static void put (String s, HttpServlet h){
        ht.put(s,  h);
    }
    static boolean contains (String s){
        return ht.containsKey(s);
    }
    static HttpServlet get(String s) {
        return ht.get(s);
    }
    static void remove(String s) {
        ht.remove(s);
    }
}
```

MyHttpServer1: Shutdown.java

```
public class Shutdown {
    static boolean flag = false;
}
```