Tutorial 1

Distributed Systems

TUM Informatik XIII
Application and Middleware Systems
https://msrg.in.tum.de/

Martin Jergler

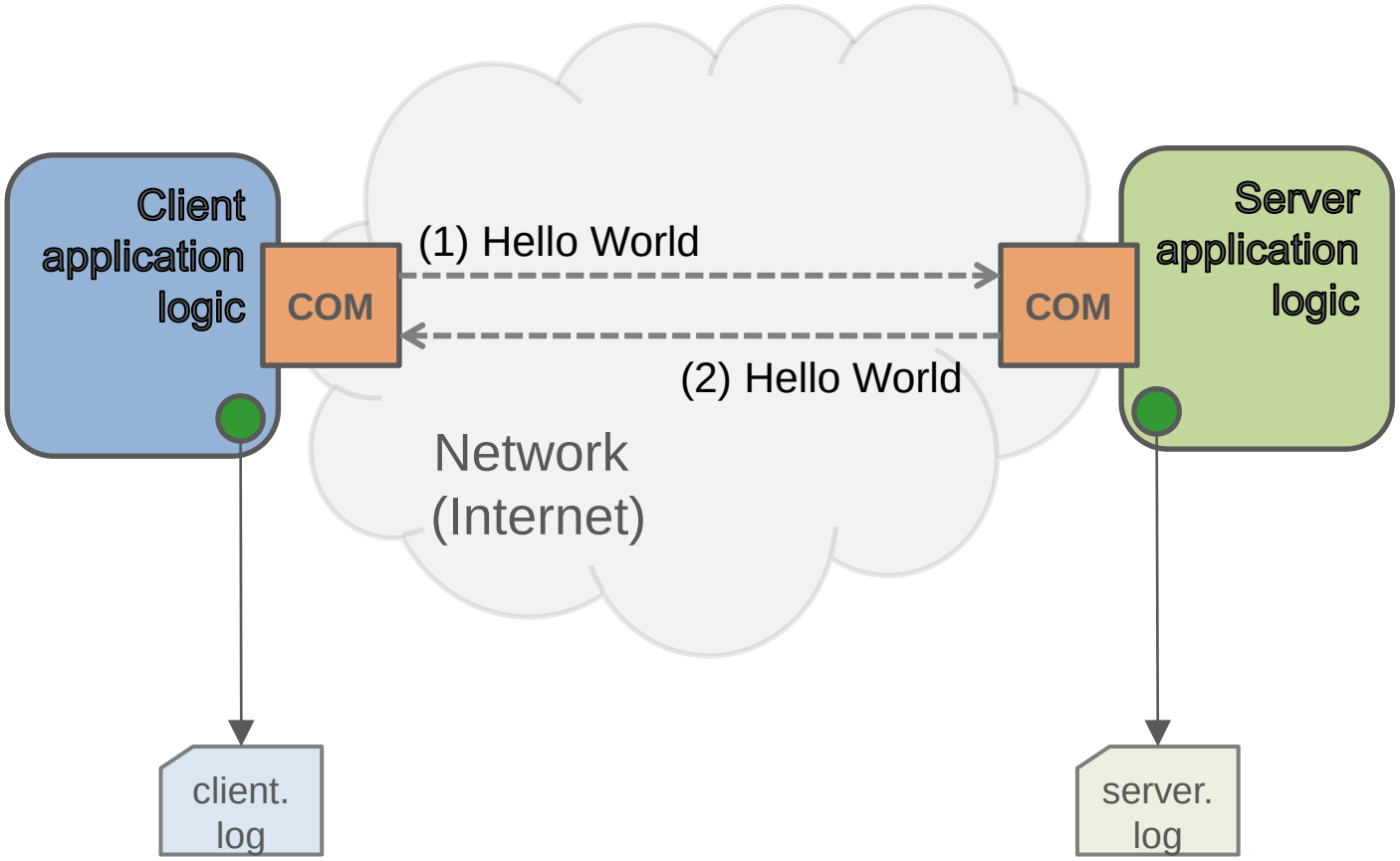
Outline

- Overview Milestone 1
- Socket programming basics
- System streams and basic shell
- Logging with log4j
- Apache Ant

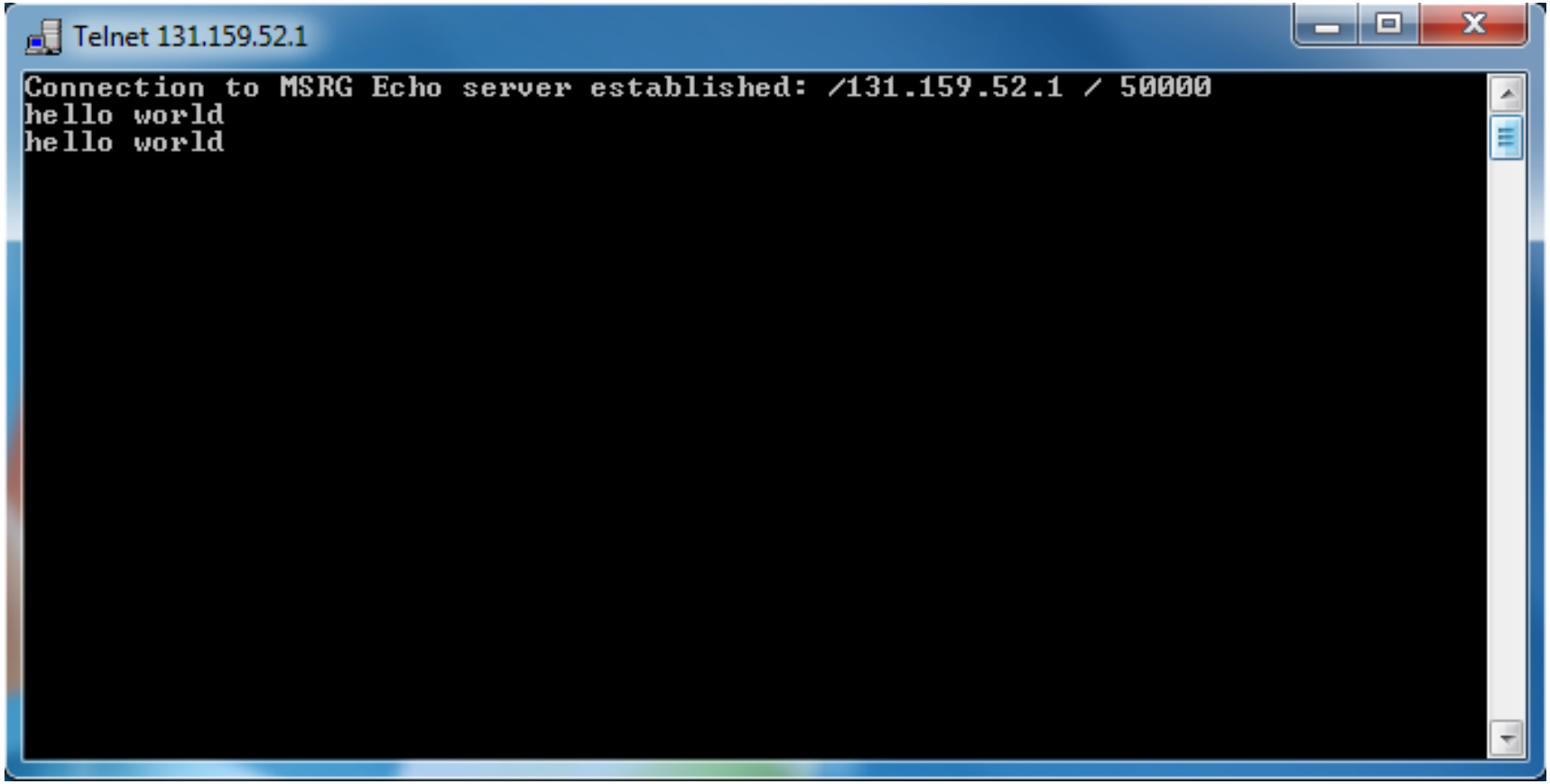
Organizational matters

- Milestone 1 online – see Link on Moodle
  - Deadline: 07.11.2012
- Register your team – see Link on Moodle
  - Important: provide me your in.tum accounts

Milestone 1 – Echo client

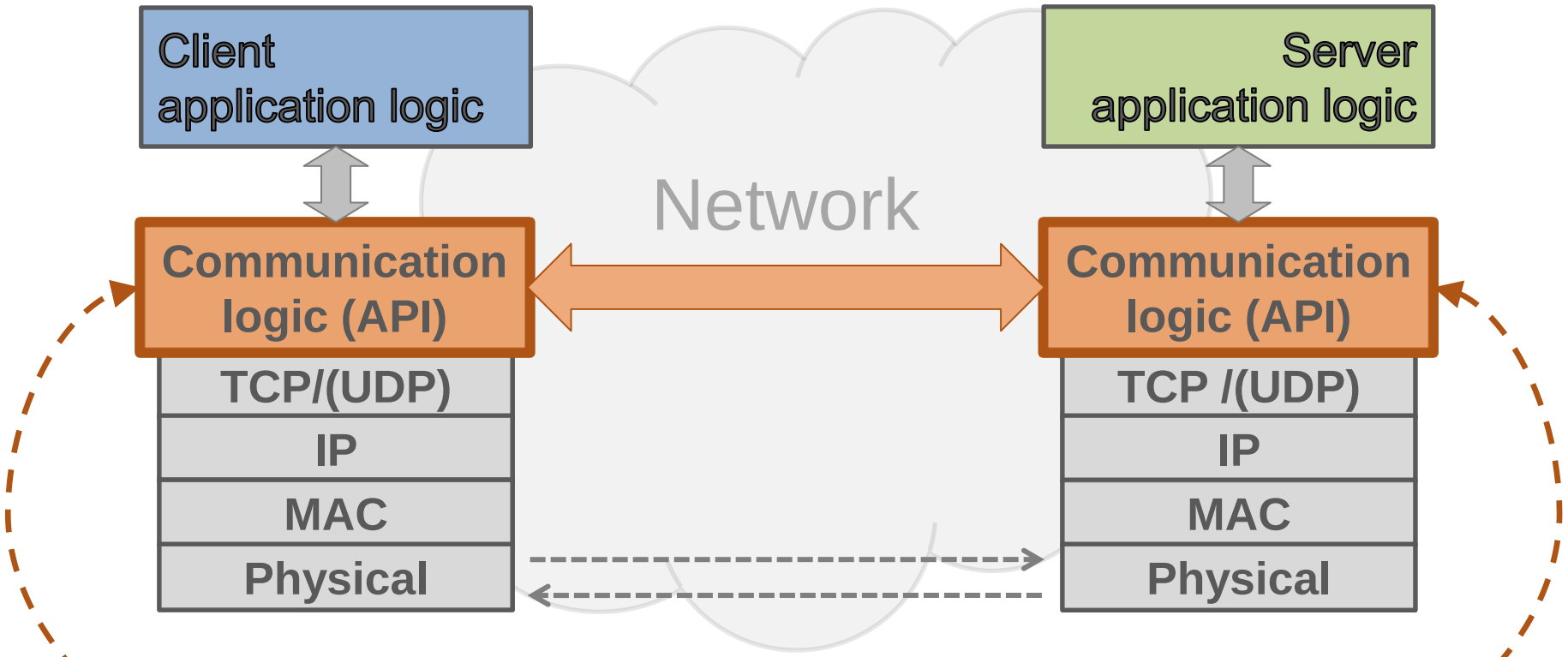


Demo - Telnet




```
>> telnet 131.159.52.1 50000
```

Communication Stack



Sockets are network programming abstractions

Outline

- Overview Milestone 1
- Socket programming basics
- System streams and basic shell
- Logging with log4j
- Apache Ant

TCP vs. UDP (Features & Guarantees)

TCP

(Transmission Control Protocol)

- connection-oriented
- reliable
- flow control
- congestion avoidance
- ordering of segments

UDP

(User Datagram Protocol)

- connection-less
- unreliable
- no flow control
- no congestion avoidance
- no inherent ordering

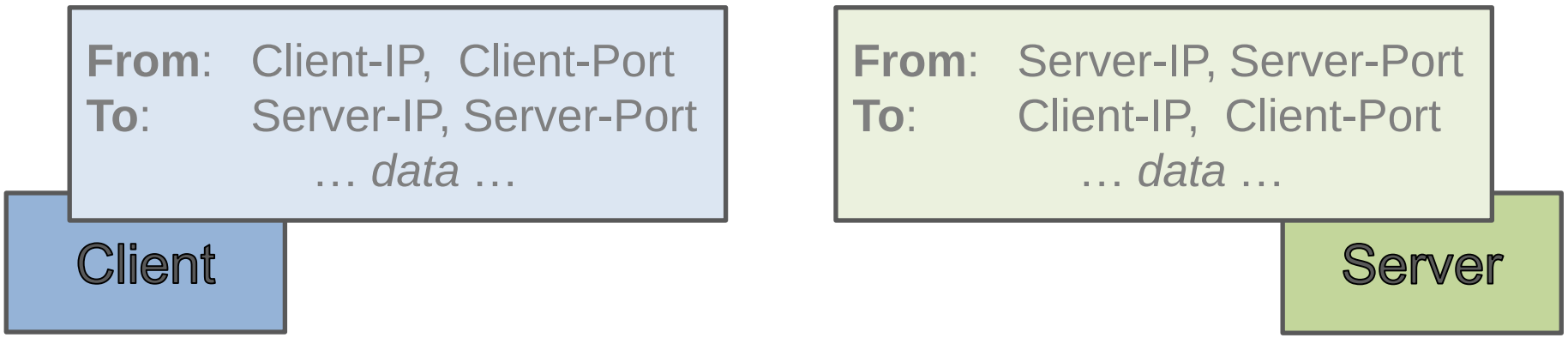
Speed ?

Sockets – general

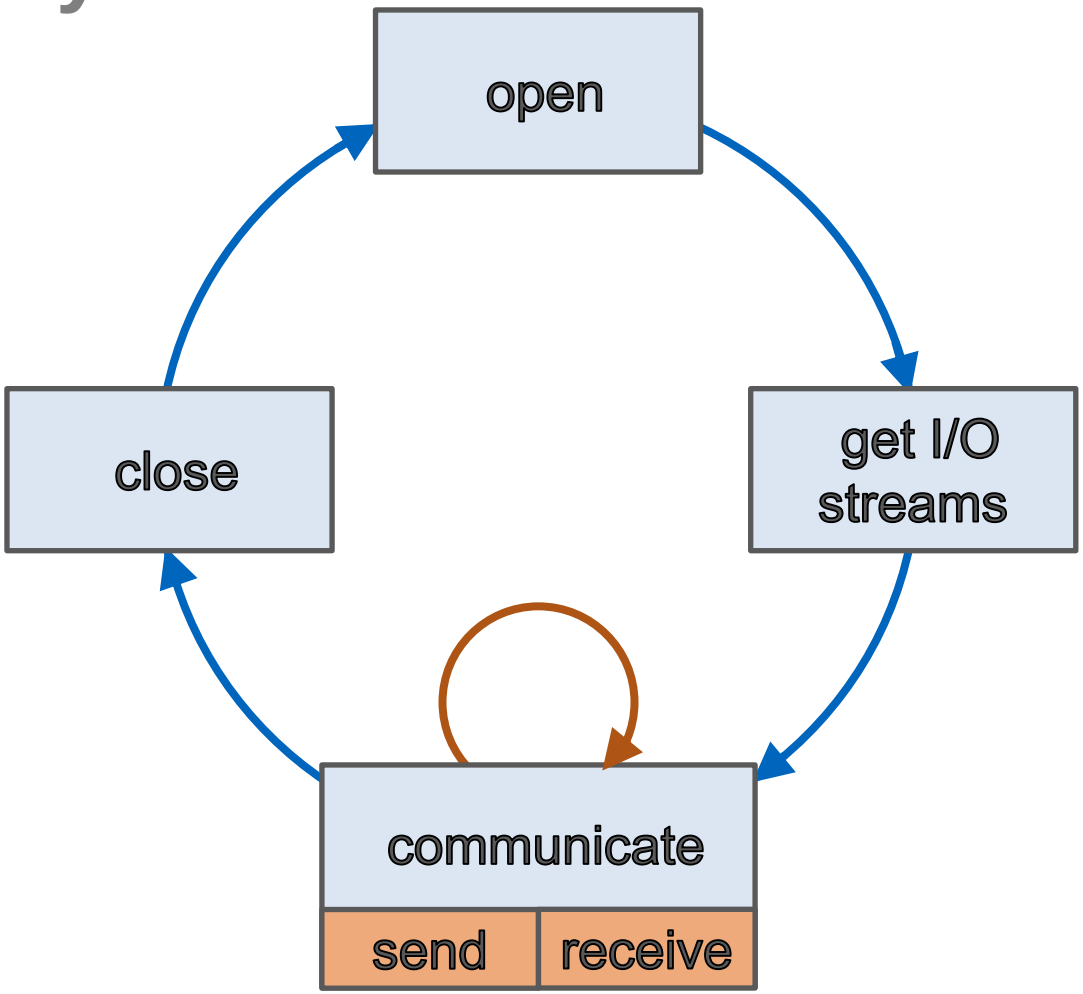
Sockets define communication endpoints!

A connection endpoint is characterized by

- IP address (host address) e.g., 131.159.52.1 (loopback: 127.0.0.1)
- Port number e.g., 50000



Socket Lifecycle



Java Sockets – client perspective

Java socket implementation: `java.net.Socket`

Create a socket and connect to (remote) host

```
public Socket(String host, int port)
```

→ may throw exception

Example

```
Socket client = new Socket(localhost, 50000);
```

API: http://docs.oracle.com/javase/6/docs/api/java/net/Socket.html

Network streams & basic communication

Sockets have an *input*- and an *output* stream

- send data via output and receive data via input

Init streams → may throw exception

```
InputStream in = client.getInputStream();
OutputStream out = client.getOutputStream();
```

Use streams → may throw exception

```
int recvByte = in.read();
out.write(sendByte);
out.flush();
```

Closing socket and streams

After the communication is done all resources have to be released.

- close streams

```
in.close();
out.close();
```

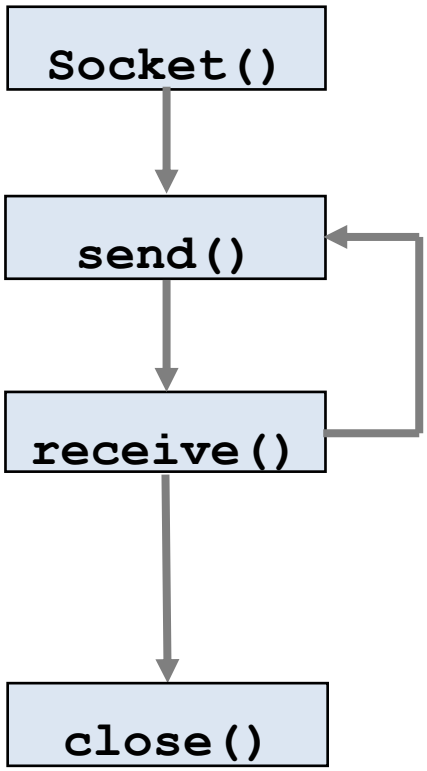
- close socket

```
client.close();
```

→ may throw exception

Putting it all together – Java template

```
try {
        /* create socket, connect to
           server and init streams */
        while (connected) {
                //communicate
        }
} catch (IOException ioe) {
        /* handle gracefully */
} finally {
        /* close streams and socket */
}
```



Exceptions

Within the lifecycle of a socket diverse Exceptions might occur.

Step	Type of exception	Reason
init socket & connect	`UnknownHostException`	Host cannot be resolved
	`IOException`	“arbitrary I/O error” → no communication possible
init streams	`IOException`	
read from in & write to out stream	`IOException`	
close streams/ socket	`IOException`	

Handle exceptions properly in your implementation!!!

Outline

- Overview Milestone 1
- Socket programming basics
- System streams and basic shell
- Logging with log4j
- Apache Ant

Shell – Command line based UI

1. Print out prompt, e.g., „`client>` “
2. Read in current line
3. Split input into separate tokens
4. Case differentiation by first token
   - Check syntactical correctness, i.e. number of parameters, format of parameters, etc.
   - Check semantic correctness
   - Execute action
5. Print out result and goto 1.

System standard streams

- `System.out` // write to console
- `System.in` // read from console

- Read line:

```
BufferedReader input;
input = new BufferedReader(new InputStreamReader(System.in));
String line = input.readLine();
```

- Write line:

```
System.out.println("some text string");
```

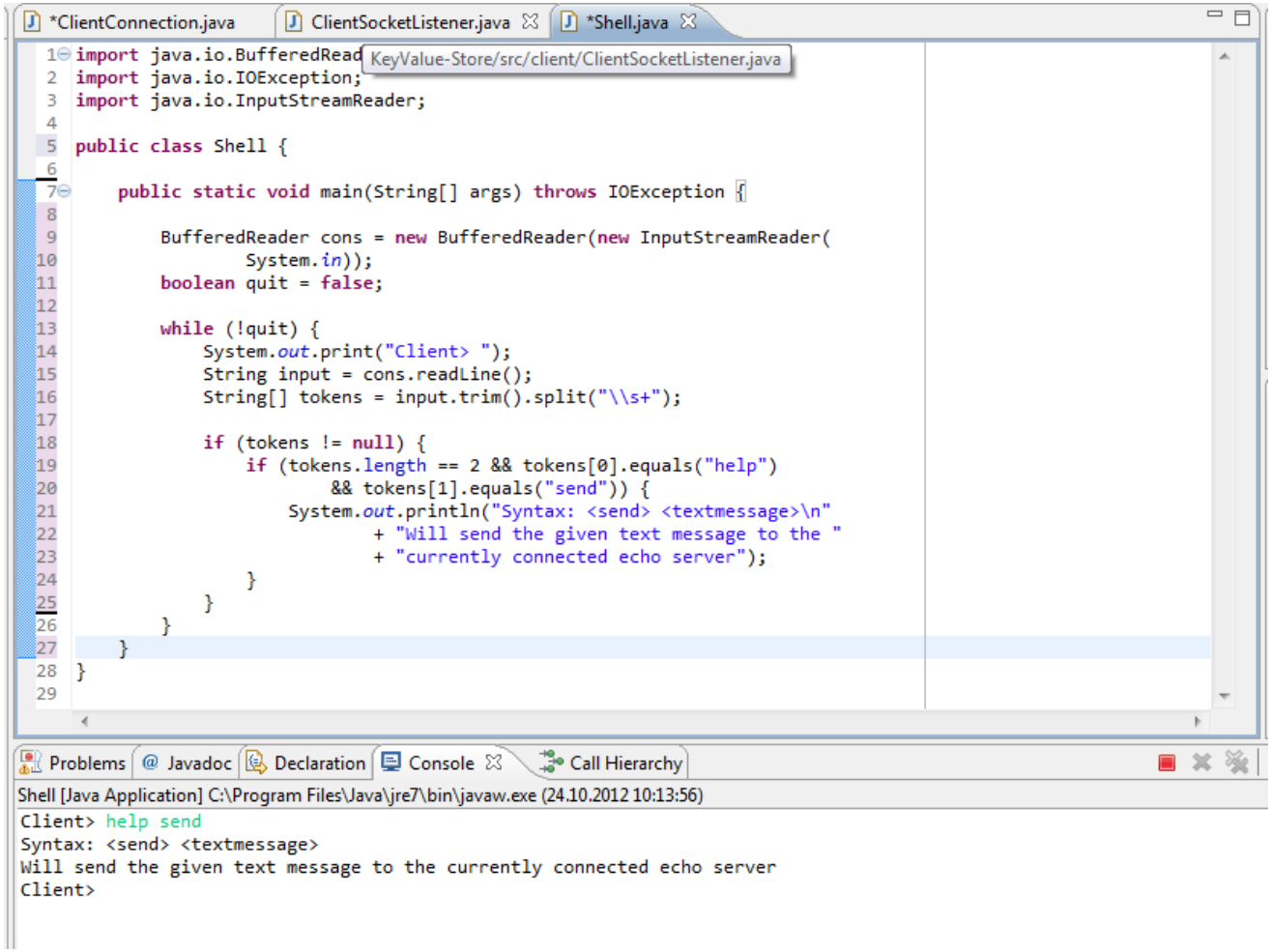
Shell template

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

   BufferedReader cons = new BufferedReader(new
               InputStreamReader(System.in));
   boolean quit = false;

   while (!quit) {
      System.out.println("Prompt> ");
      String input = cons.readLine();
      String[] tokens = input.trim().split("\\s+");
   }
}
```

Demo shell example – "help" command



Outline

- Overview Milestone 1
- Socket programming basics
- System streams and basic shell
- Logging with log4j
- Apache Ant

Logging

- What is logging ?
- Why do we need logging ?
- How do we implement logging ?

~~System.out.println(…);~~

Download & further info: http://logging.apache.org/log4j/1.2/

Logging with Log4j - Features

Centrally configurable, dynamic logging framework

- Hierarchical naming => Rootlogger is top
- Different granularities => log levels
  (ALL | DEBUG | INFO | WARN | ERROR | FATAL | OFF)
- Different targets => log appenders
  (console, file, DailyRollingFile, email, socket, telnet, JDBC, etc)
- Customize logs => log layouts
  (SimpleLayout, PatternLayout, HTMLLayout and XMLLayout)
- Dynamical control => configuration files

Setup logging with log4j

1. add log4j jar to build path, class path and

```
import org.apache.log4j.Logger;
```

2. Initialize logger and configure it
   - create static logger

```
Logger mylogger = Logger.getLogger(App.class);
```

   - customize your logger
     - set log level:

```
mylogger.setLevel(Level.ALL);
```

     - configure and add appenders: (simple & just to console)

```
SimpleLayout sL = new SimpleLayout();
ConsoleAppender cA = new ConsoleAppender(sL);
mylogger.addAppender(cA);
```

Example: FileAppender with Pattern layout

```
// initialize logger
Logger mylogger = Logger.getLogger(App.class);
String logDir = “logging/client.log”;
String pattern = "%d{ISO8601} %-5p [%t] %c: %m%n“;

PatternLayout pLayout = new PatternLayout(pattern);
FileAppender fa = new FileAppender(pLayout, logDir, true );
mylogger.addAppender(fa);

// use logger
mylogger.info(“my first log”);

// log
2012-10-24 15:46:38,367 INFO  [main] Shell: my first log
```

Example: .config file

log.config

```
log4j.rootLogger=ALL, fa

log4j.appender.fa=org.apache.log4j.FileAppender
log4j.appender.fa.File=logging/client.log
log4j.appender.fa.layout=org.apache.log4j.PatternLayout
log4j.appender.fa.layout.ConversionPattern
        =%d{ISO8601} %-5p [%t] %c: %m%n
```

```
// use log.config
Logger mylogger = Logger.getLogger(App.class);
PropertyConfigurator.configure("logging/log.config");
```

Outline

- Overview Milestone 1
- Socket programming basics
- System streams and basic shell
- Logging with log4j
- Apache Ant

Apache Ant – Overview

Ant is a java tool for automatically

- building, and
- executing

Java programs

Download

- similar to “make”
- requires JRE
- project is defined and controlled by “build.xml”

  <project> is root element

```
<project basedir="." default="build-jar" name="Echo Client">
```

- Specific actions are defined within targets

Apache Ant – Tasks

Frequent tasks:

- *javac* – compile source code
- *mkdir* – create directory
- *delete* – delete directory
- *zip* – compress files
- ….

It is possible to define your own tasks by implementing them in java. However, the standard set is sufficient in most cases.

Apache Ant – Targets

Targets define actions and may depend on other targets.

Typical targets: (cf. “our” build.xml)

- init – setup environment, create directories
- build – include libraries and compile source code
- build jar – create a jar file
- run – execute the application
- clean – remove files/ directories from last build/run

Apache Ant – Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project basedir="." default="build-jar" name="Echo Client">

  <property name="debuglevel" value="source,lines,vars"/>
  <property name="target" value="1.6"/>
  <property name="source" value="1.6"/>

  <property name="build.dir"     value="bin"/>
  <property name="src.dir"       value="src"/>
  <property name="lib.dir"       value="libs"/>
  <property name="jar.file"      value="echoClient.jar"/>
  <property name="manifest.file" value="MANIFEST.MF"/>

  <!-- The main class of your application should be named according to the
     following -->
  <property name="main.class" value="ui.Application"/>

  <path id="external.jars">
    <fileset dir="${lib.dir}" includes="**/*.jar"/>
  </path>

  <path id="project.classpath">
    <pathelement location="${src.dir}"/>
    <path refid="external.jars" />
  </path>

  <target name="init">
    <mkdir dir="${build.dir}"/>
    <copy includeemptydirs="false" todir="${build.dir}">
      <fileset dir="${src.dir}">
        <exclude name="**/*.launch"/>
```

Ant tutorial: http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

Questions ?

https://msrg.in.tum.de/

Martin Jergler | martin.jergler@tum.de | 01.06.058