

UNIT 6

Socket for Clients

(5 Hours) — Network Programming (CAC355)

Disclaimer: These notes were generated automatically from the CACS355 syllabus and may contain errors or omissions. Please verify important details (syntax, method names, default values) against your textbook (Java Network Programming, E. R. Harold) and class notes before the exam.

6.1 Introduction to Socket

A **socket** is one end point of a two-way communication link between two programs on a network. It is identified by an **IP address + port number**. A **port** is a 16-bit number (0–65535) that selects a particular application on a host; ports 0–1023 are well-known (reserved), 1024–49151 are registered and the rest are dynamic.

Java's `java.net.Socket` class provides a **TCP** client socket. TCP gives a reliable, ordered, byte-stream connection. A connection is fully identified by the 4-tuple (client IP, client port, server IP, server port).

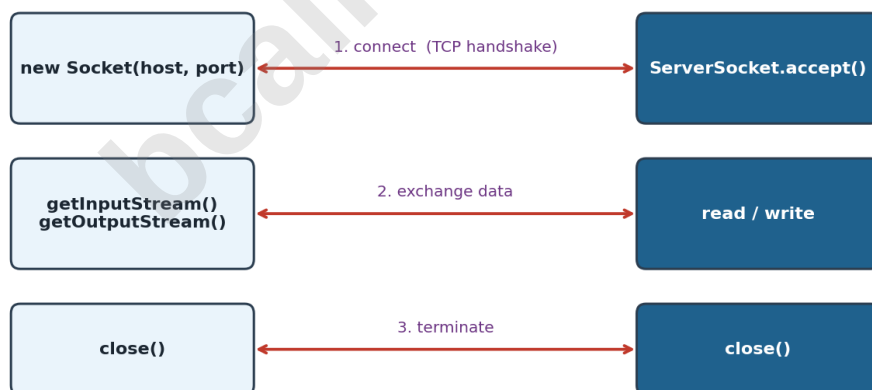


Figure 6.1 — Life cycle of a TCP client and server socket

6.2 Using Sockets

Investigating protocols with telnet

telnet is a simple tool for opening a TCP connection to any port and typing commands by hand. It is useful for learning text-based protocols (HTTP, SMTP, POP3, daytime, echo) before writing the Java client.

```
$ telnet www.example.com 80
GET / HTTP/1.1
Host: www.example.com
<blank line>
```

Reading from servers with sockets

Steps: (1) create the `Socket`, (2) get its `InputStream` and wrap it in a reader, (3) read until end of stream (read returns `-1`) and (4) close. Use `try-with-resources` so the socket is closed automatically. Example: the **daytime** service (port 13).

```
try (Socket s = new Socket("time.nist.gov", 13)) {
    s.setSoTimeout(15000);
    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()), "US-ASCII");
    String line;
    while ((line = in.readLine()) != null) System.out.println(line);
} catch (IOException e) { e.printStackTrace(); }
```

Listing 6.1 — Reading from a server

Writing to servers with sockets

Get the `OutputStream`, wrap it (e.g., in a `Writer` or `PrintWriter`) and write. Call **`flush()`** so buffered data is actually sent. Many text protocols end lines with `\r\n`. Reading and writing can be combined for request–response protocols such as an echo or dict client.

```
try (Socket s = new Socket("localhost", 7)) { // echo service
    Writer out = new OutputStreamWriter(s.getOutputStream(), "UTF-8");
    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()), "UTF-8");
    out.write("Hello\r\n");
    out.flush();
    System.out.println(in.readLine());
}
```

Listing 6.2 — Writing then reading

Note: Half-close: `shutdownOutput()` tells the server there is no more data from the client while the client can still read the reply.

6.3 Constructing and Connecting Sockets

Basic constructors

Constructor	Description
<code>Socket(String host, int port)</code>	Resolves the host name and connects; throws <code>UnknownHostException</code> , <code>IOException</code>

Constructor	Description
<code>Socket(InetAddress host, int port)</code>	Connects to a resolved address (no DNS lookup)

If the host refuses the connection you get **ConnectException** (connection refused); if it cannot be reached at all, **NoRouteToHostException** or a timeout. The constructor blocks until the connection is made, so it may wait a long time.

Picking a local interface to connect from

A machine with several network interfaces can choose the local address and port:

- **`Socket(String host, int port, InetAddress localAddr, int localPort)`**
- **`Socket(InetAddress host, int port, InetAddress localAddr, int localPort)`**
- `localPort = 0` means “choose any free port”.

Constructing without connecting

`new Socket()` creates an unconnected socket so that options can be set first (e.g., before connect for `SO_RCVBUF`). Connect later with **`connect(SocketAddress)`** or **`connect(SocketAddress, int timeoutMillis)`** — the latter gives a **connection timeout**, which the ordinary constructors cannot do.

```
Socket s = new Socket();
s.connect(new InetSocketAddress("www.example.com", 80), 5000); // 5 s timeout
```

Socket addresses

`SocketAddress` is an abstract class; the TCP/IP implementation is `InetSocketAddress` (host + port). Constructors: *`InetSocketAddress(InetAddress, int)`*, *`(String host, int port)`* (resolves), *`createUnresolved(host, port)`* (does not resolve). Methods: `getAddress()`, `getHostName()`, `getPort()`, `isUnresolved()`. A connected socket returns *`getRemoteSocketAddress()`* and *`getLocalSocketAddress()`*.

Proxy servers

A socket can be made to go through a proxy: `new Socket(Proxy)` creates an unconnected socket that will use the given proxy (typically a **SOCKS** proxy: *`new Proxy(Proxy.Type.SOCKS, addr)`*). HTTP proxies are not supported by plain sockets. System properties `socksProxyHost` and `socksProxyPort` set a proxy for all sockets.

6.4 Getting Information about a Socket

Method	Returns
<code>getInetAddress()</code>	Remote address
<code>getPort()</code>	Remote port
<code>getLocalAddress()</code>	Local address
<code>getLocalPort()</code>	Local port (chosen automatically if not specified)
<code>getRemoteSocketAddress()</code> / <code>getLocalSocketAddress()</code>	Address + port objects
<code>getInputStream()</code> / <code>getOutputStream()</code>	The data streams

Closed or connected?

Method	Meaning
<code>isClosed()</code>	true after <code>close()</code> has been called
<code>isConnected()</code>	true if the socket ever connected (stays true after closing)
<code>isBound()</code>	true if bound to a local port
<code>isInputShutdown()</code> / <code>isOutputShutdown()</code>	Whether each half has been shut down

Note: To test that a socket is currently open and connected use `isConnected()` && `!isClosed()`.

`toString()`

Returns text such as `Socket[addr=www.example.com/93.184.216.34,port=80,localport=52341]`; useful for debugging and logging.

Closing

`close()` releases the socket; always close in a *finally* block or use try-with-resources. Closing the socket's input or output stream also closes the socket.

6.5 Setting Socket Options

Option	Methods	Effect
<code>TCP_NODELAY</code>	<code>setTcpNoDelay(boolean)</code> / <code>getTcpNoDelay()</code>	true disables Nagle's algorithm (send small packets immediately); good for interactive, low-latency use

Option	Methods	Effect
SO_LINGER	setSoLinger(boolean, int seconds)	How long close() blocks to send remaining data; 0 = discard data and reset immediately
SO_TIMEOUT	setSoTimeout(int ms)	Read timeout; read() throws SocketTimeoutException after the delay (0 = infinite)
SO_RCVBUF / SO_SNDBUF	setReceiveBufferSize(int) / setSendBufferSize(int)	Size of the TCP receive/send buffers (a hint to the OS)
SO_KEEPALIVE	setKeepAlive(boolean)	Periodic probe on idle connections (about every 2 hours) to detect a dead peer
OOBINLINE	setOOBInline(boolean)	Urgent (out-of-band) data received inline with normal data; otherwise discarded; sent with sendUrgentData(int)
SO_REUSEADDR	setReuseAddress(boolean)	Allows binding to an address/port still in TIME_WAIT
IP_TOS	setTrafficClass(int)	Type-of-service / class of service bits: low cost (0x02), high reliability (0x04), high throughput (0x08), low delay (0x10)

6.6 Sockets in GUI Applications

Network calls block, so a GUI (Swing) must never do them in the event dispatch thread — the window would freeze. Do the socket work in a **background thread** (a Thread or Swing's **SwingWorker**) and update the GUI with *SwingUtilities.invokeLater()* or SwingWorker's *done()/publish()*.

Whois

Whois (port 43) is a simple query protocol for domain-registration data. The client connects to a whois server (e.g., whois.internic.net), sends the domain name followed by `\r\n`, reads the reply until the server closes the connection, and displays it. A GUI version has a text field, a button and a text area filled by the worker thread.

```
try (Socket s = new Socket("whois.internic.net", 43)) {
    Writer out = new OutputStreamWriter(s.getOutputStream(), "US-ASCII");
    out.write("example.com\r\n");
    out.flush();
    BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream(), "US-ASCII"));
    String line;
    while ((line = in.readLine()) != null) System.out.println(line);
}
```

Listing 6.3 — Core of a Whois client

A network client library

To reuse networking code, separate it from the GUI in a **client library** class: it hides the socket details, exposes methods like *lookup(String)*, throws meaningful exceptions and can be used from console and GUI programs alike. This follows the principle of separating the *protocol logic* from the *user interface*.

Key Takeaways

- ✓ A socket = IP address + port; `java.net.Socket` is a TCP client socket.
- ✓ Client steps: create/connect, get streams, read/write (flush!), close (try-with-resources).
- ✓ Constructors: (host, port), (`InetAddress`, port), with local address, or unconnected + `connect(addr, timeout)`.
- ✓ Info methods: `getInetAddress`, `getPort`, `getLocalPort`, `isConnected`, `isClosed`, `toString`.
- ✓ Options: `TCP_NODELAY`, `SO_LINGER`, `SO_TIMEOUT`, `SO_RCVBUF`/`SO_SNDBUF`, `SO_KEEPALIVE`, `OOBINLINE`, `SO_REUSEADDR`, `IP_TOS`.
- ✓ In GUIs, run network code in background threads; Whois is the classic example.

— End of Unit 6 —