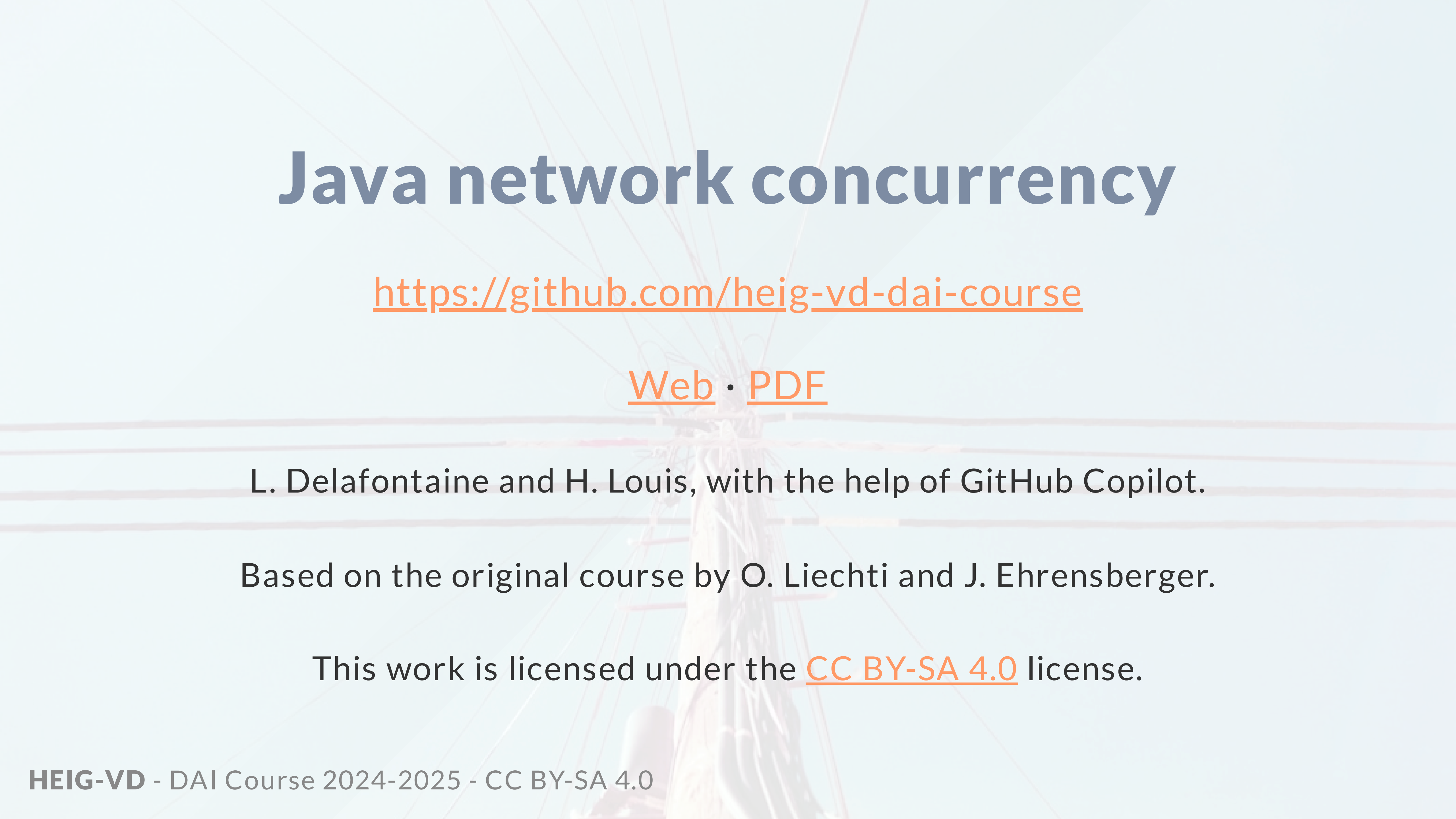


Java network concurrency

https://github.com/heig-vd-dai-course

[Web](#) · [PDF](#)

L. Delafontaine and H. Louis, with the help of GitHub Copilot.

Based on the original course by O. Liechti and J. Ehrensberger.

Objectives

- Program your own TCP client/server applications in Java with the Socket API
- Understand how to handle multiple clients at the same time
- Understand how to process data from streams

Your applications will be able to communicate over the network!

TCP

More details for this section in the course material. You can find other resources and alternatives as well.

TCP

TCP is a transport protocol that is similar to a phone call:

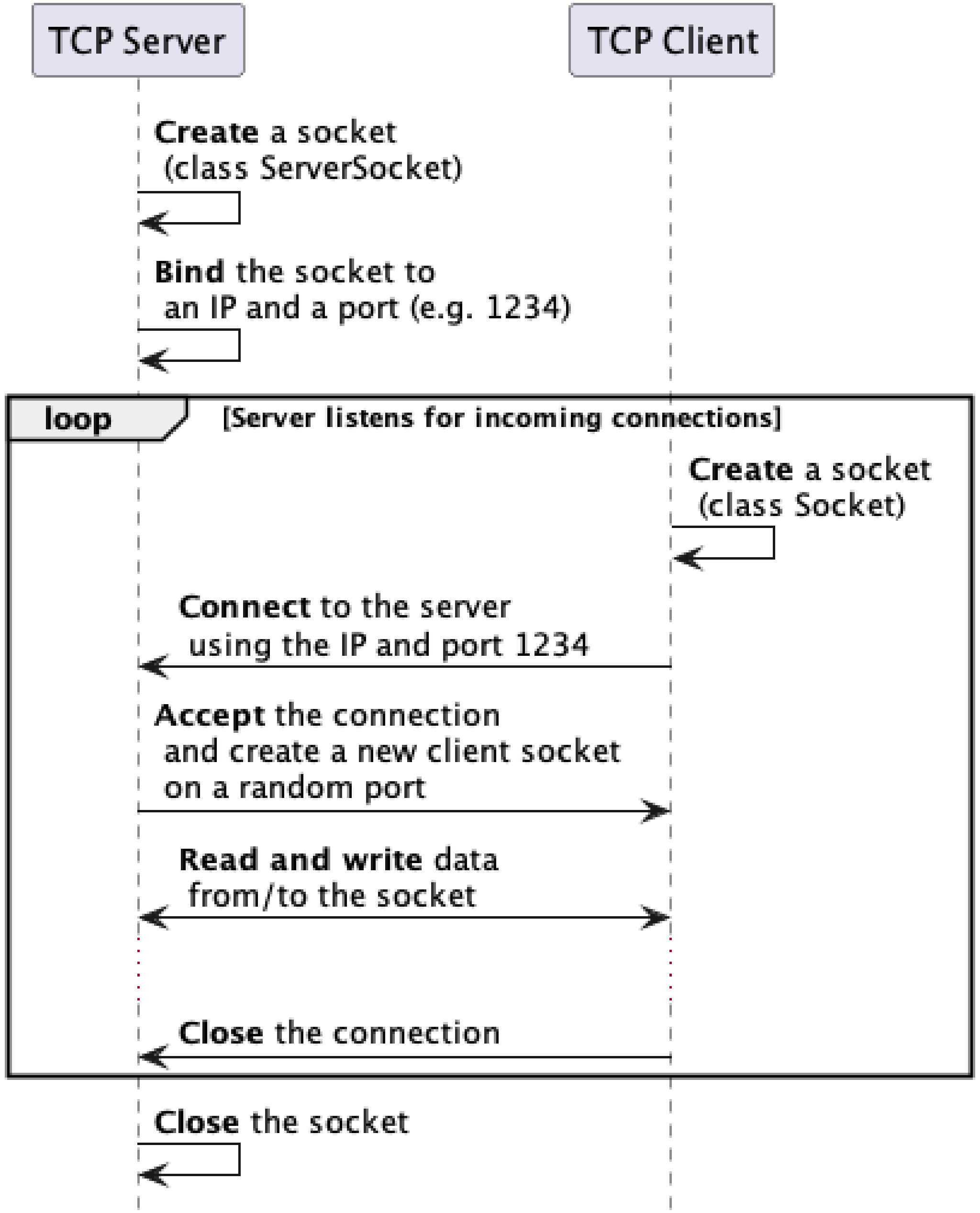
1. A connection is established between two parties
2. Data sent is guaranteed to arrive in the same order
3. Data can be sent again

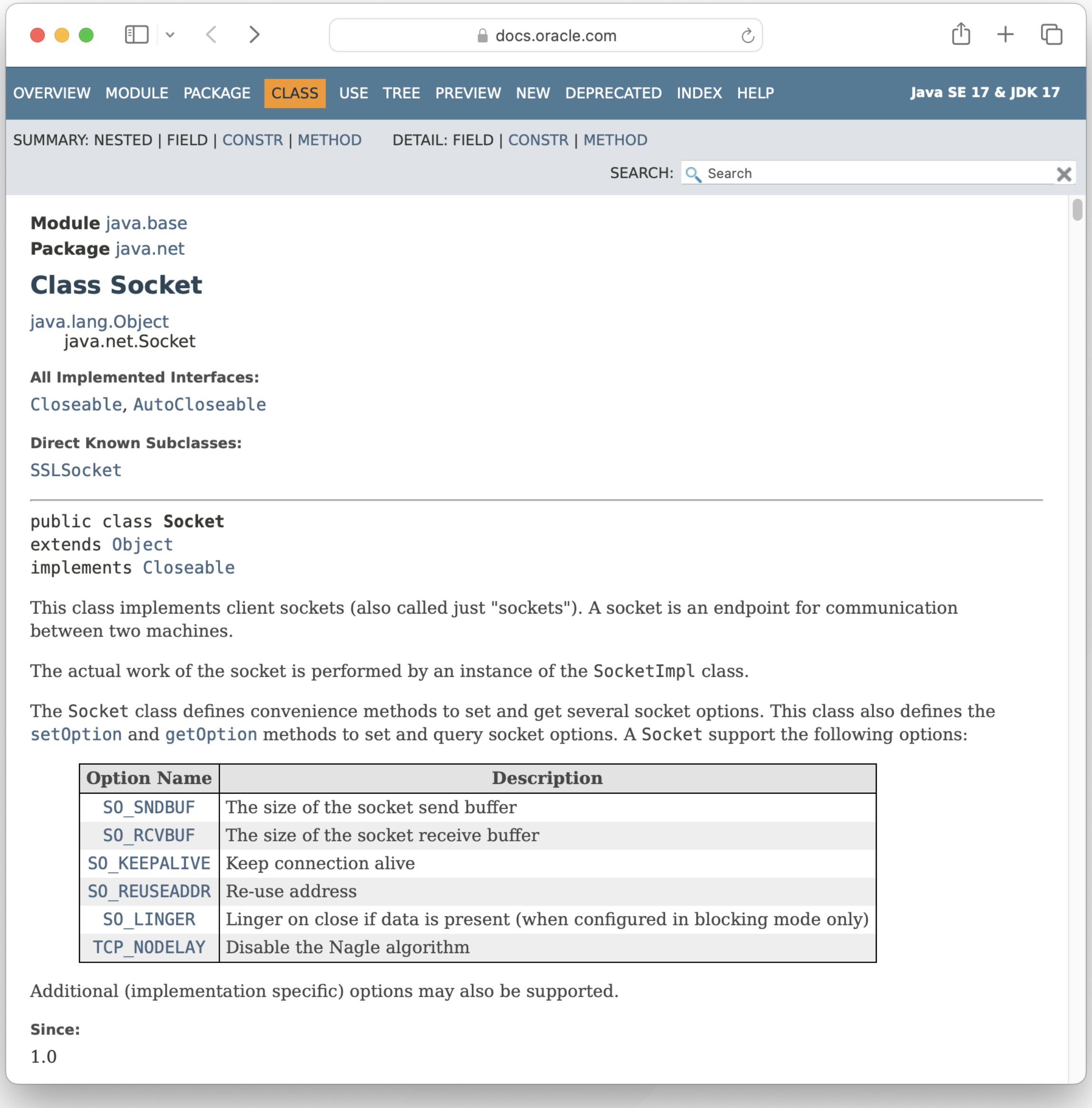
The Socket API

More details for this section in the course material. You can find other resources and alternatives as well.

The Socket API

- Originally developed by Berkeley University
- Ported to Java and many other languages
- Provides a simple API to use TCP and UDP
- A socket is a connection between two parties using a protocol and a port


TCP Server
TCP Client
Create a socket (class ServerSocket)
Bind the socket to an IP and a port (e.g. 1234)
loop
[Server listens for incoming connections]
Create a socket (class Socket)
Connect to the server using the IP and port 1234
Accept the connection and create a new client socket on a random port
Read and write data from/to the socket
Close the connection
Close the socket


docs.oracle.com
OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 17 & JDK 17
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH: Search

Module java.base
Package java.net

Class Socket

java.lang.Object
    java.net.Socket

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

SSLSocket

```
public class Socket
extends Object
implements Closeable
```

This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.

The actual work of the socket is performed by an instance of the SocketImpl class.

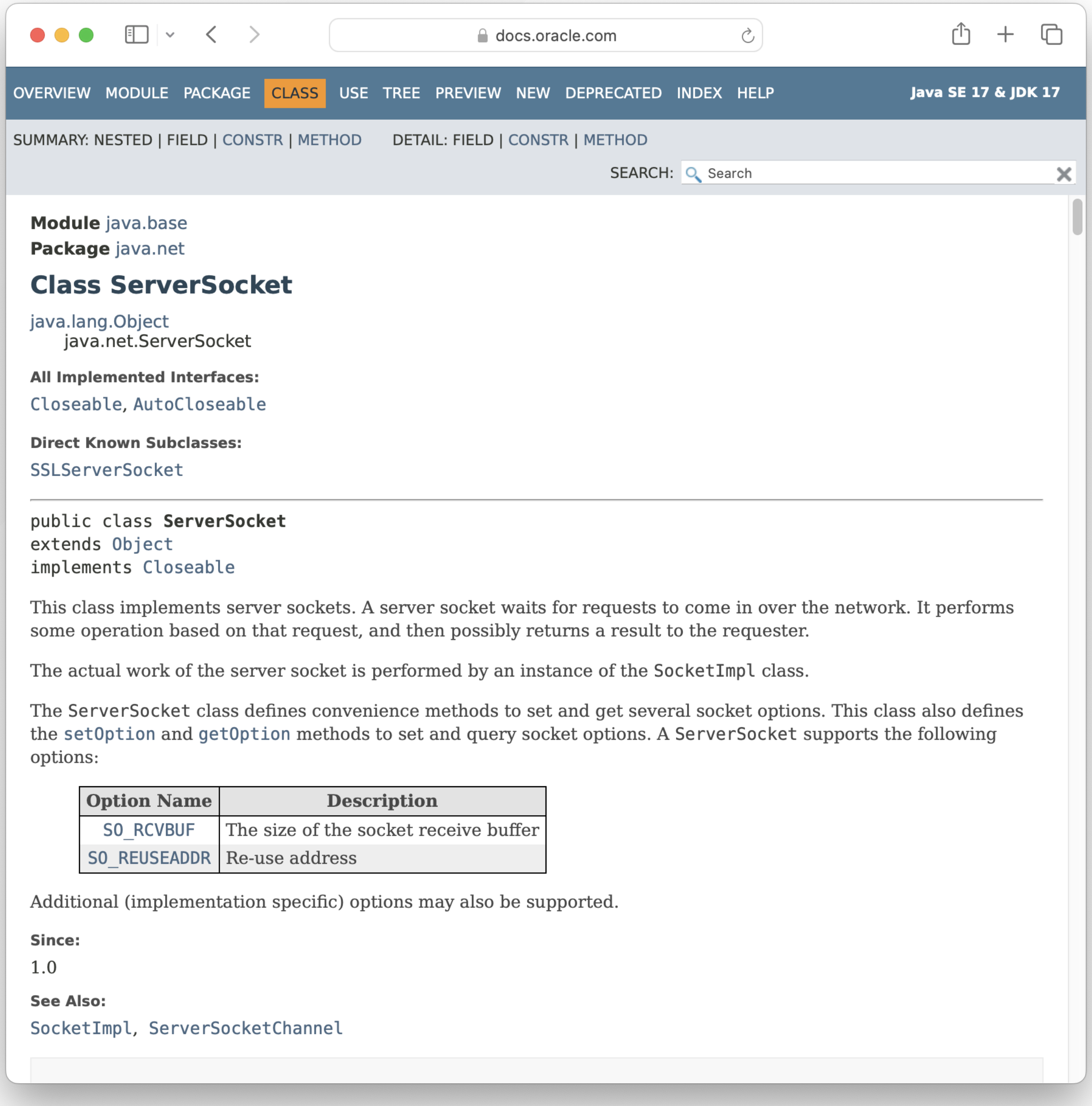
The Socket class defines convenience methods to set and get several socket options. This class also defines the setOption and getOption methods to set and query socket options. A Socket support the following options:

Option Name	Description
SO_SNDBUF	The size of the socket send buffer
SO_RCVBUF	The size of the socket receive buffer
SO_KEEPALIVE	Keep connection alive
SO_REUSEADDR	Re-use address
SO_LINGER	Linger on close if data is present (when configured in blocking mode only)
TCP_NODELAY	Disable the Nagle algorithm

Additional (implementation specific) options may also be supported.

Since:

1.0


docs.oracle.com
OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP
Java SE 17 & JDK 17
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD
SEARCH: Search

Module java.base
Package java.net

Class ServerSocket

java.lang.Object
    java.net.ServerSocket

All Implemented Interfaces:

Closeable, AutoCloseable

Direct Known Subclasses:

SSLServerSocket

```
public class ServerSocket
extends Object
implements Closeable
```

This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

The actual work of the server socket is performed by an instance of the SocketImpl class.

The ServerSocket class defines convenience methods to set and get several socket options. This class also defines the setOption and getOption methods to set and query socket options. A ServerSocket supports the following options:

Option Name	Description
SO_RCVBUF	The size of the socket receive buffer
SO_REUSEADDR	Re-use address

Additional (implementation specific) options may also be supported.

Since:

1.0

See Also:

SocketImpl, ServerSocketChannel

Client/server common functions

Operation	Description
`socket()`	Creates a new socket
`getInputStream()`	Gets the input stream of a socket
`getOutputStream()`	Gets the output stream of a socket
`close()`	Closes a socket

Client structure and functions

1. Create a `Socket`
2. Connect the socket to an IP address and a port number
3. Read and write data from/to the socket
4. Flush and close the socket

Operation	Description
`connect()`	Connects a socket to an IP address and a port number

Server structure and functions

1. Create a `ServerSocket`
2. Bind the socket to an IP address and a port number
3. Listen for incoming connections
4. Loop
    1. Accept an incoming connection - creates a new `Socket` on a port
    2. Read and write data from/to the socket
    3. Flush and close the socket
5. Close the `ServerSocket`

Operation	Description
`bind()`	Binds a socket to an IP address and a port number
`listen()`	Listens for incoming connections
`accept()`	Accepts an incoming connection

To make it simple, a socket is just like a file that you can open, read from, write to and close. To exchange data, sockets on both sides must be connected.

Processing data from streams

More details for this section in the course material. You can find other resources and alternatives as well.

Processing data from streams

- Sockets use data streams to send and receive data, just like files
- Get an input stream to read data from a socket
- Get an output stream to write data to a socket

Variable length data

Data sent can have a variable length. Manage this using one of the two methods:

- Use a delimiter
- Communicate a fixed length

This must be defined by your application protocol!

Using a delimiter:

```java
// End of transmission character
String EOT = "\u0004";

// Read data until the delimiter is found
String line;
while ((line = in.readLine()) != null && !line.equals(EOT)) {
  System.out.println(
    "[Server " + SERVER_ID + "] received data from client: " + line
  );
}
```

Communicating a fixed length:

```java
// Send the length of the data
out.write("DATA_LENGTH " + data.length() + "\n");

// Send the data
out.write(data);
```

```java
// Read the length of the data
String[] parts = in.readLine().split(" ");
int dataLength = Integer.parseInt(parts[1]);

// Read the data
for (int i = 0; i < dataLength; i++) {
  System.out.print((char) in.read());
}
```

Handling one client at a time

More details for this section in the course material. You can find other resources and alternatives as well.

Handling one client at a time

1. Create a socket to listen to incoming connections
2. Create a new socket for the client
3. Handle the connection

Analogy: a restaurant with one table

➡️ Simple but quite useless...

Handling multiple clients at the same time

More details for this section in the course material. You can find other resources and alternatives as well.

Handling multiple clients at the same time

Handle multiple clients at the same time is called concurrency.

Concurrency can be achieved with:

- Multi-processing
- Multi-threading
- Asynchronous programming

Multi-processing

- Create an entirely new process for each client
- Heavyweight and slow
- Not recommended
- Analogy: a new restaurant for each customer, including the kitchen, waiters, etc.

Multi-threading

- Create a new thread for each client
- More lightweight and faster than multi-processing
- Recommended with thread pool to limit resource usage
- Not enough or too many threads can slow down the system
- Analogy: a new or limited number of tables for each customer

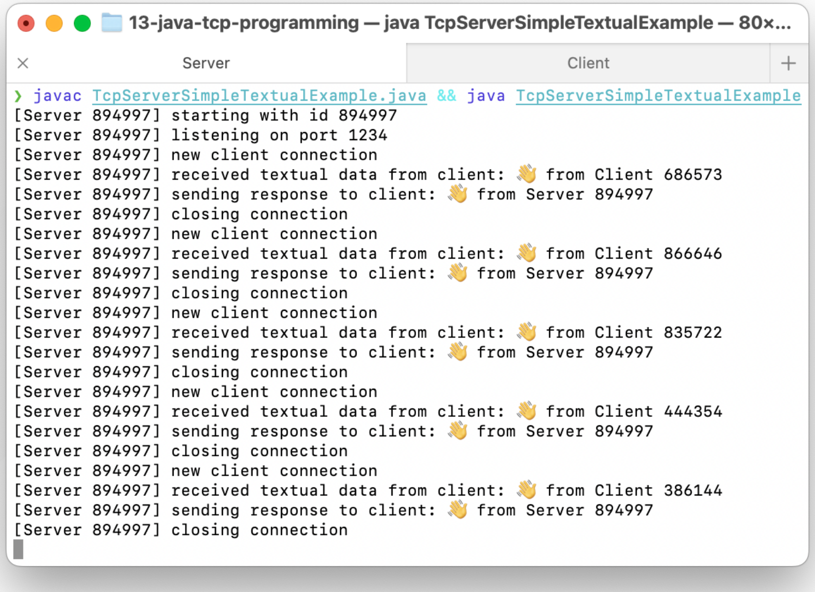
Asynchronous programming

- Handle multiple clients with a single thread
- Very performant!
- Analogy: a food truck - you get a ticket, wait, get your food and leave
- Out of scope for this course but interesting to know! Node.js is a good example for this approach

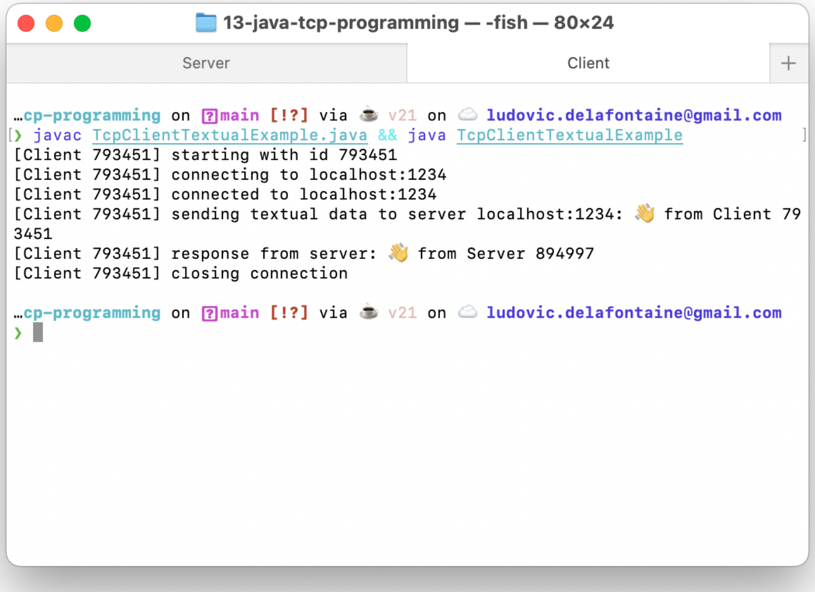
Practical content

What will you do?

- Send an email using a SMTP client written in Java with the Socket API
- Run full client/server examples and understand how concurrent clients are handled


13-java-tcp-programming — java TcpServerSimpleTextualExample — 80×...
Server
Client

```
❯ javac TcpServerSimpleTextualExample.java && java TcpServerSimpleTextualExample
[Server 894997] starting with id 894997
[Server 894997] listening on port 1234
[Server 894997] new client connection
[Server 894997] received textual data from client: 👋 from Client 686573
[Server 894997] sending response to client: 👋 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 👋 from Client 866646
[Server 894997] sending response to client: 👋 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 👋 from Client 835722
[Server 894997] sending response to client: 👋 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 👋 from Client 444354
[Server 894997] sending response to client: 👋 from Server 894997
[Server 894997] closing connection
[Server 894997] new client connection
[Server 894997] received textual data from client: 👋 from Client 386144
[Server 894997] sending response to client: 👋 from Server 894997
[Server 894997] closing connection
```



```
…cp-programming on ?main [!?] via ☕ v21 on ☁ ludovic.delafontaine@gmail.com
❯ javac TcpClientTextualExample.java && java TcpClientTextualExample
[Client 793451] starting with id 793451
[Client 793451] connecting to localhost:1234
[Client 793451] connected to localhost:1234
[Client 793451] sending textual data to server localhost:1234: 👋 from Client 79
3451
[Client 793451] response from server: 👋 from Server 894997
[Client 793451] closing connection

…cp-programming on ?main [!?] via ☕ v21 on ☁ ludovic.delafontaine@gmail.com
❯
```

Find the practical content

You can find the practical content for this chapter on GitHub.

Finished? Was it easy? Was it hard?

Can you let us know what was easy and what was difficult for you during this chapter?

This will help us to improve the course and adapt the content to your needs. If we notice some difficulties, we will come back to you to help you.

➡️ GitHub Discussions

You can use reactions to express your opinion on a comment!

What will you do next?

You will start the practical work!

Sources

- Main illustration by Carl Nenzen Loven on Unsplash
- Illustration by Aline de Nadai on Unsplash
- Illustration by Alexander Andrews on Unsplash
- Illustration by Oleksandra Bardash on Unsplash
- Illustration by patricia serna on Unsplash
- Illustration by Elliott Stallion on Unsplash
- Illustration by K8 on Unsplash
- Illustration by bckfwd on Unsplash

- Illustration by James Elchico on Unsplash
- Illustration by micheile henderson on Unsplash