

A blue-toned world map with glowing nodes and connecting lines, overlaid with binary code (0s and 1s) and technical diagrams, representing a global network.

Distributed Systems

Practicum introduction



MOSAIC

Modeling Of Systems And Internet Communication
University of Antwerp

Practical considerations

Coordinates

- Tom De Schepper – M.G.215
- tom.deschepper@uantwerpen.be

Practicum

- 1 project assignment
- Groups of 2
- 1 final deadline before the start of the exams

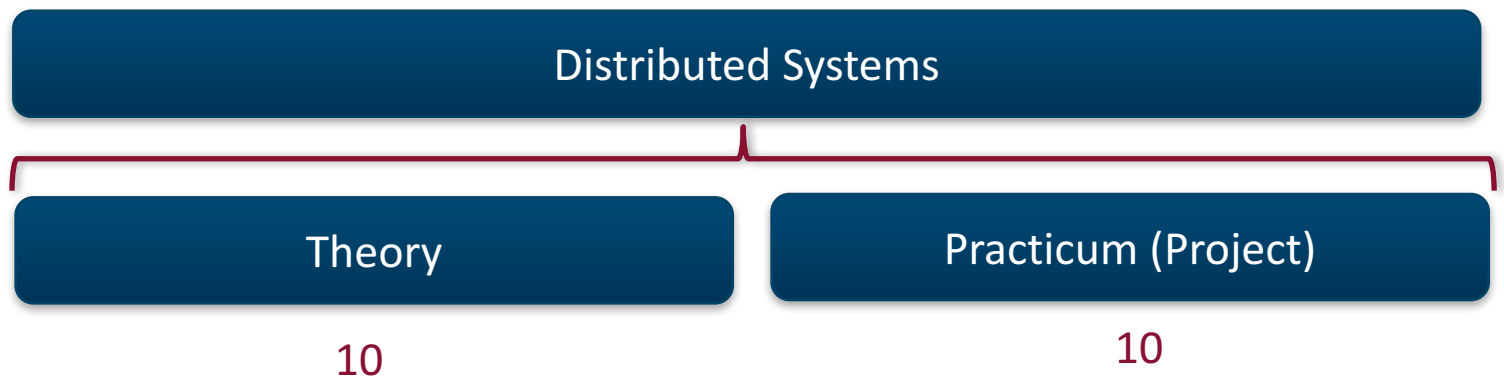
Schedule

Date		Topic
13/10	09u00 – 12u00	Project assignment
20/10	09u00 – 12u00	Q&A project
17/11	09u00 – 12u00	Q&A project
8/12	09u00 – 12u00	Q&A project
22/12	09u00 – 12u00	Q&A project
8/01	23u59	Deadline: Project

Practical considerations

Grades

- 10 points (out of 20)
- Need 50 % for both theory and project



Programming

Tools

- Java as programming language
- Eclipse
- Avro library
- Other libraries are allowed

Programming

Java vs C++

- Compilation:
 - C++: to system-level bytecode, runs natively
 - Java: to JVM bytecode, runs on Java Virtual Machine
- Error behaviour java is defined
 - E.g., out-of-bound-index,
- Similar language syntax
- Object Oriented
- Memory management
 - auto garbage collection, no explicit pointers, ...

Programming

Java vs C++

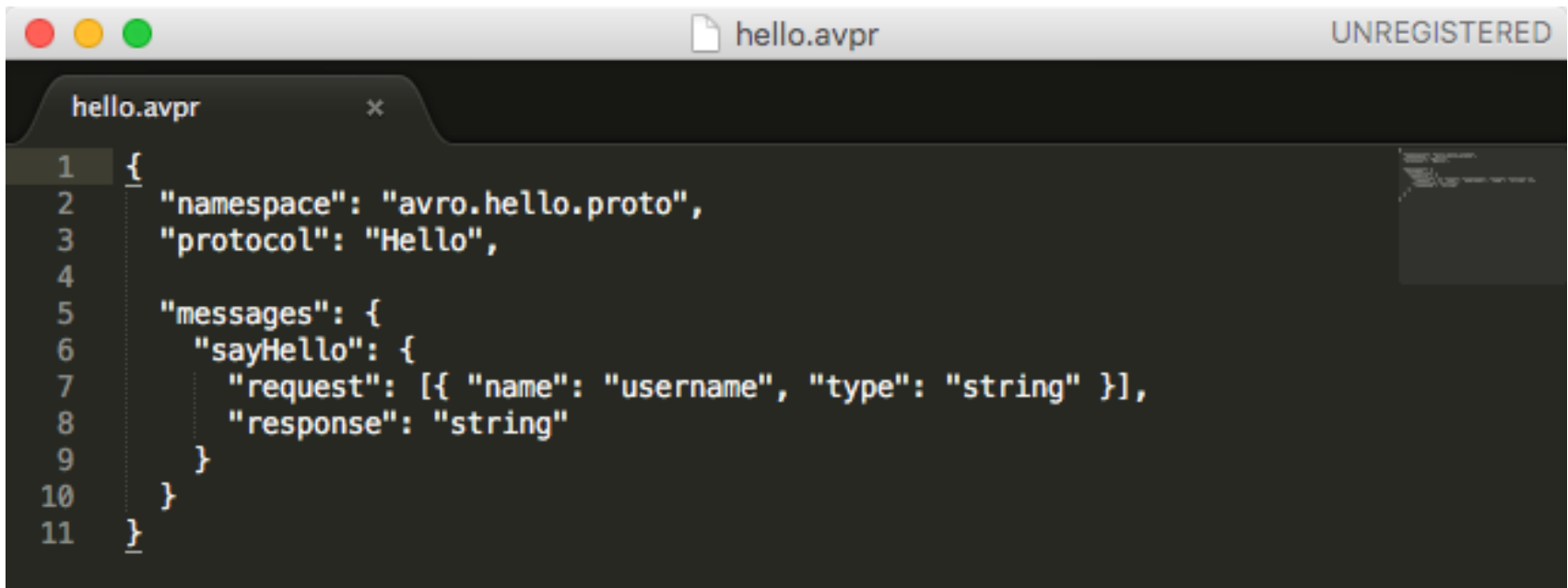
- Java is always passed-by-value
 - Primitive types and object references
- Other
 - No operation overloading
 - Final instead of const
 - I/O with streams
 - Interface
 - Java serialization
 - ...

Apache Avro

- Remote procedure call and data serialization framework
- Newer than Java RMI
- Schema based (JSON)



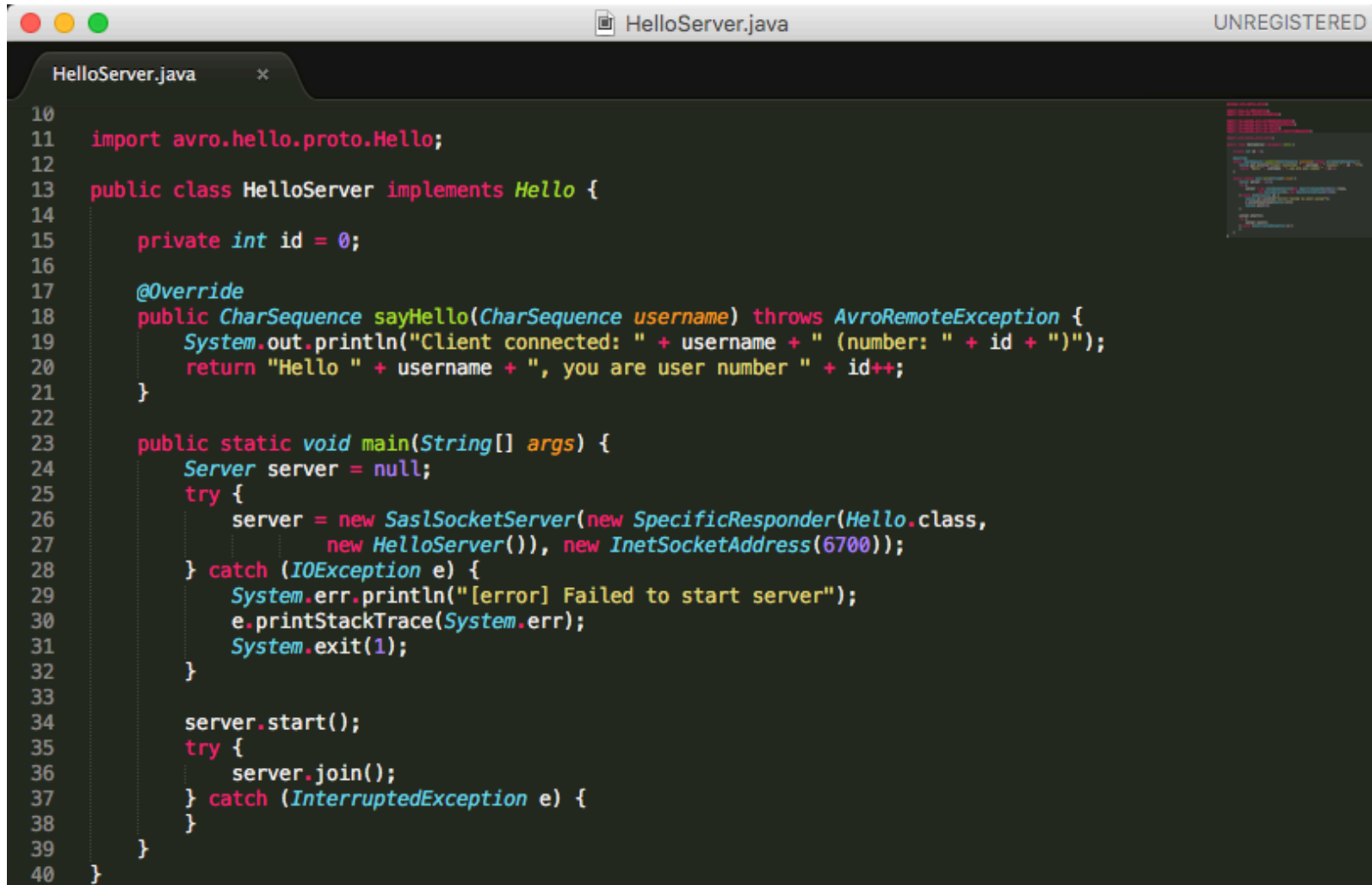
Schema



A screenshot of a code editor window titled 'hello.avpr' with a tab labeled 'hello.avpr' and a close button. The editor shows an Avro schema in JSON format. The schema defines a namespace 'avro.hello.proto', a protocol 'Hello', and a message 'sayHello' with a 'request' field of type 'string' and a 'response' field of type 'string'. The code is as follows:

```
1 {  
2   "namespace": "avro.hello.proto",  
3   "protocol": "Hello",  
4  
5   "messages": {  
6     "sayHello": {  
7       "request": [{ "name": "username", "type": "string" }],  
8       "response": "string"  
9     }  
10  }  
11 }
```

Server



```
10
11 import avro.hello.proto.Hello;
12
13 public class HelloServer implements Hello {
14
15     private int id = 0;
16
17     @Override
18     public CharSequence sayHello(CharSequence username) throws AvroRemoteException {
19         System.out.println("Client connected: " + username + " (number: " + id + ")");
20         return "Hello " + username + ", you are user number " + id++;
21     }
22
23     public static void main(String[] args) {
24         Server server = null;
25         try {
26             server = new SaslSocketServer(new SpecificResponder(Hello.class,
27                 new HelloServer()), new InetSocketAddress(6700));
28         } catch (IOException e) {
29             System.err.println("[error] Failed to start server");
30             e.printStackTrace(System.err);
31             System.exit(1);
32         }
33
34         server.start();
35         try {
36             server.join();
37         } catch (InterruptedException e) {
38         }
39     }
40 }
```

Client



```
9
10 import avro.hello.proto.Hello;
11
12 public class HelloClient {
13
14     public static void main(String[] args) {
15         try {
16             Transceiver client = new SaslSocketTransceiver(new InetSocketAddress(6700));
17             Hello proxy = (Hello) SpecificRequestor.getClient(Hello.class, client);
18             CharSequence response = proxy.sayHello("Bob");
19             System.out.println(response);
20             client.close();
21         } catch (IOException e) {
22             System.err.println("Error connecting to server...");
23             e.printStackTrace(System.err);
24             System.exit(1);
25         }
26     }
27 }
28
```