UNIKASSEL VERSITÄT

Research Group Software Engineering

Programmiermethodik

Übung 13

Sommersemester 2010

Fachgebiet Software Engineering

Andreas Scharf

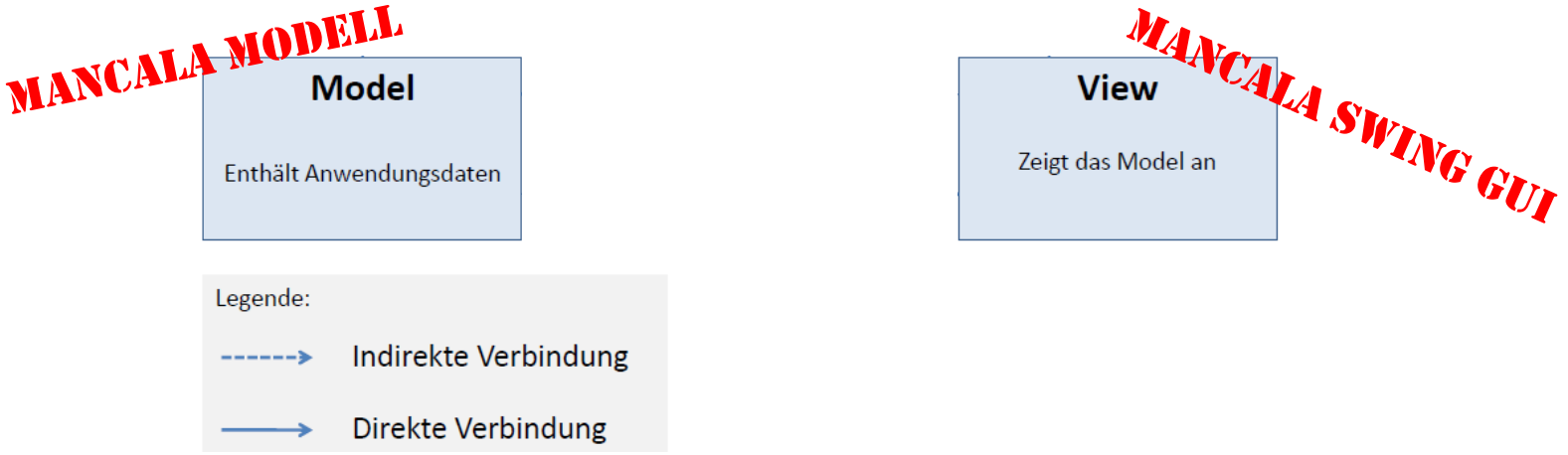
andreas.scharf@cs.uni-kassel.de

Agenda

- Vorstellung Musterlösung HA9
- Mancala Showroom
- Client/Server Kommunikation in Java
- Vorstellung HA11 (Zusatzaufgabe)
- Praktische Übung

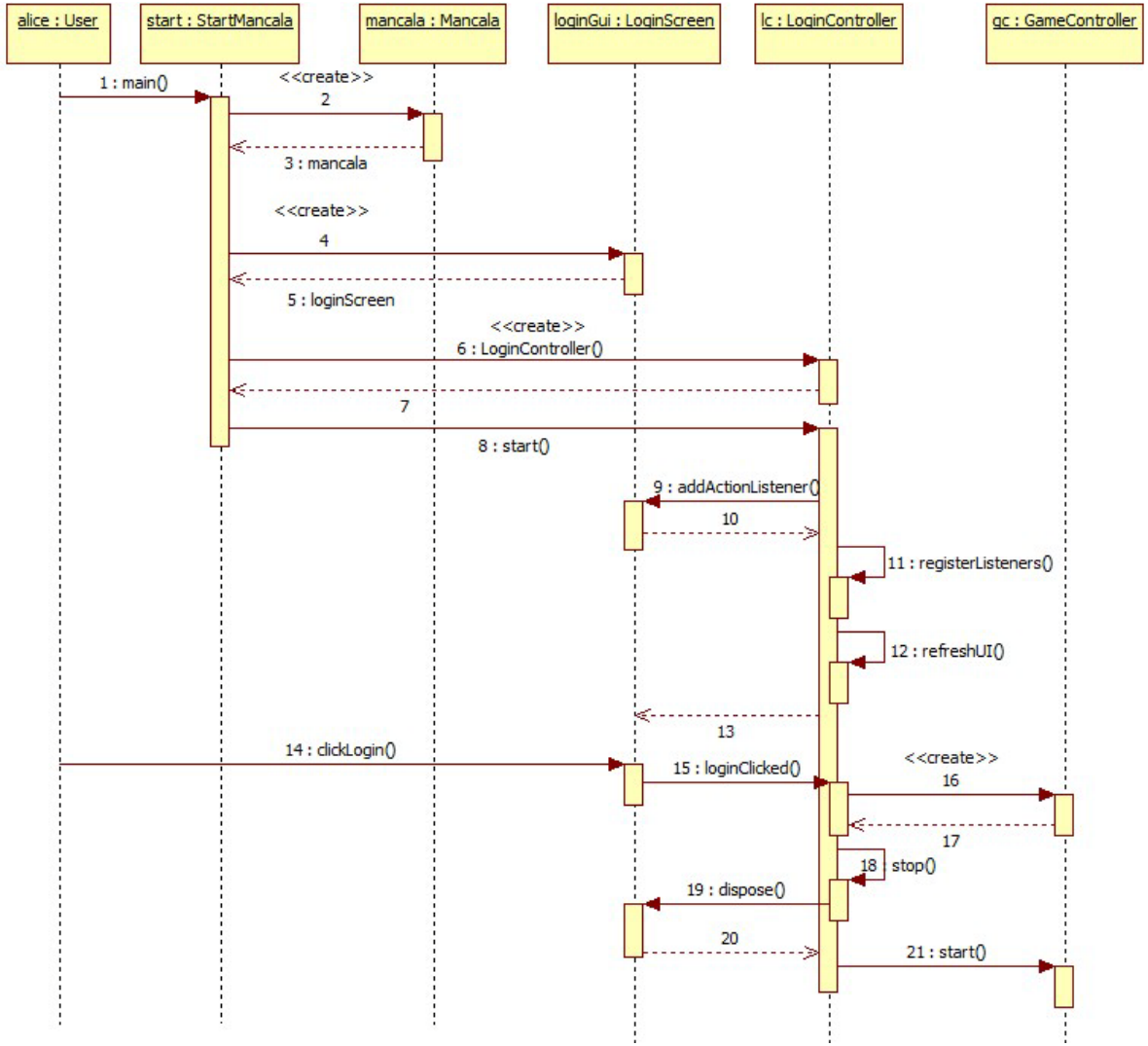
Vorstellung Musterlösung HA9 I

- **Controller synchronisieren Modell und Anzeige**



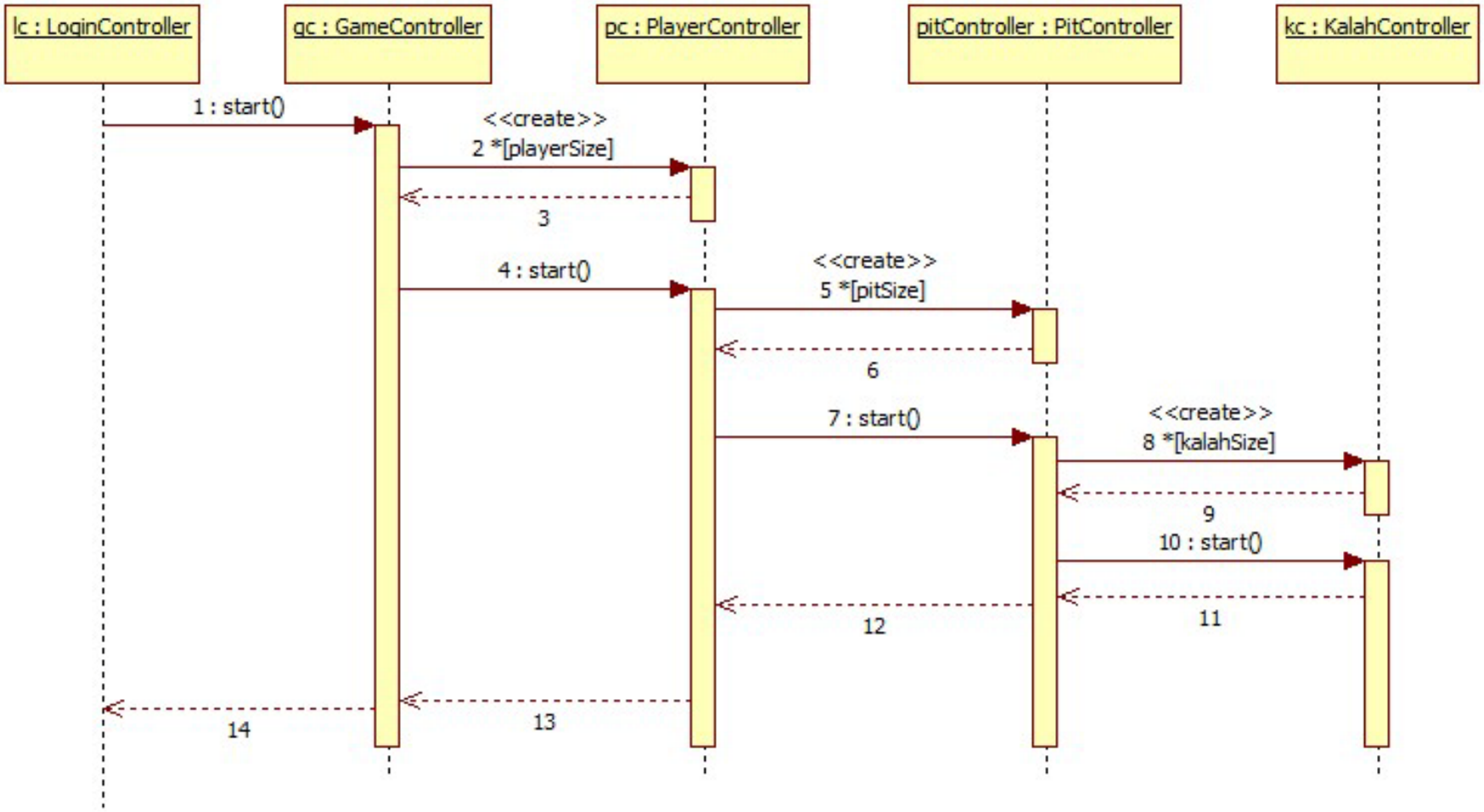
- **Idealfall: Controller lassen sich löschen -> keine Compilefehler!**

Vorstellung Musterlösung HA9 II



Sequenzdiagramm beim Starten des Spiels

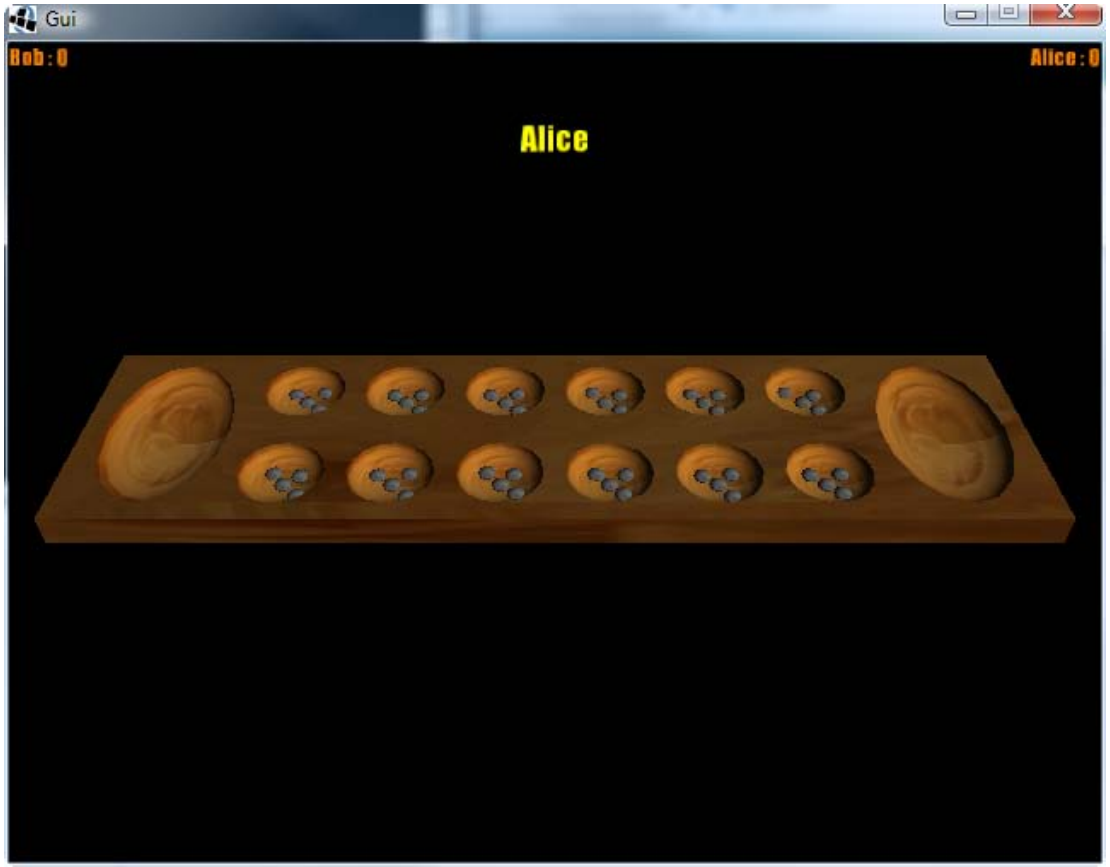
Vorstellung Musterlösung HA9 III



Sequenzdiagramm beim Starten des Spiels

Mancala Showroom

- **Sehr viele schöne Lösungen**



Client/Server Kommunikation in Java I

- **Netzwerkprogrammierung ist grundsätzlich nicht ganz einfach:**
  - Verschiedene Protokolle wie TCP/IP, UDP
  - Mehrbenutzerfähigkeit: Ein Server soll mehrere Verbindungen entgegennehmen können
  - Es gibt Bücker, die sich ausschließlich mit (Teilen) der Netzwerkprogrammierung auseinandersetzen
  - …
- **Java abstrahiert von der zugrunde liegenden Übermittlungsschicht**
  - Sockets
  - Streams (Input- und OutputStreams)

Client/Server Kommunikation in Java II

- **Sockets**
  - Sind eine plattformunabhängige, standardisierte Schnittstelle
  - Bidirektionale Verbindung zwischen zwei Programmen
  - Verbinden Anwendungen auf verschiedenen, jedoch häufig auf dem selben Rechner
  - Unterscheidung in Stream Sockets (TCP) und Datagram Sockets (UDP)
- **Verbindung wird definiert durch**
  - Serveradresse (z.B. `localhost`, `192.168.0.4`, …)
  - Port (1-65535)

Client/Server Kommunikation in Java III

- **Socket Programmierung in Java ist vergleichsweise einfach**
- **Beispiel: Aufbau einer Verbindung zu einem Server**

```
public static void main(String[] args)
   {
      Socket socket = null;
      DataInputStream in = null;

      try
      {
         socket = new Socket("localhost", 80);

         in = new DataInputStream(socket.getInputStream());

         String incomingMessage = in.readUTF();
         System.out.println(incomingMessage);
      }
      catch (UnknownHostException e)
      {
         System.err.println("Don't know about host: localhost.");
      }
      catch (IOException e)
      {
         System.err.println("Couldn't get I/O for "
                            + "the connection to: localhost.");
      }
   }
```


Socket öffnen


Stream holen


Daten lesen

Client/Server Kommunikation in Java IV

- **Statt 1x lesen, so lange lesen wie es geht:**

```
public static void main(String[] args)
   {
      Socket socket = null;
      DataInputStream in = null;

      try
      {
         socket = new Socket("localhost", 80);

         in = new DataInputStream(socket.getInputStream());
         while(true)
         {
            String incomingMessage = in.readUTF();
            System.out.println(incomingMessage);
         }
      }
      catch (UnknownHostException e)
      {
         System.err.println("Don't know about host: localhost.");
      }
      catch (IOException e)
      {
         System.err.println("Couldn't get I/O for "
                            + "the connection to: localhost.");
      }
   }
```

Client/Server Kommunikation in Java V

- **Entgegennehmen von Verbindungen:**

```
public static void main(String[] args)
   {
      ServerSocket serverSocket = null;
      Socket client = null;

      try
      {
         serverSocket = new ServerSocket(5000);
         client = serverSocket.accept();
         DataInputStream input =
                     new DataInputStream(client.getInputStream());
         DataOutputStream output =
                     new DataOutputStream(client.getOutputStream());

         while(true)
         {
            // Send/receive client messages
            String message = input.readUTF();
            System.out.println(message);
            output.writeUTF("Echo: " +message);
         }
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
```

ServerSocket öffnen und auf port 5000 hören

Lesen

Schreiben

Client/Server Kommunikation in Java VI

- **Problem: Server müssen mit mehreren Verbindungen umgehen können, Aufrufe wie**

```
client = serverSocket.accept();
```

oder

```
// Send/receive client messages
String message = input.readUTF();
```

sind blockierend!

- **Lösung: Für jede Verbindung einen neuen Thread aufmachen!**

Client/Server Kommunikation in Java VII

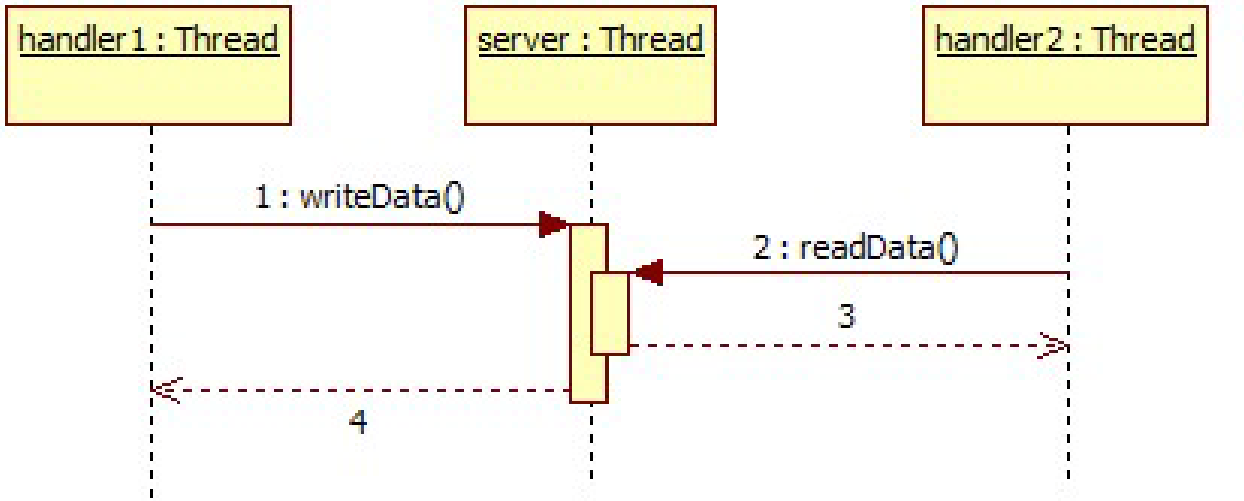
- **Für jede eingehende Verbindung einen ConnectionHandler erstellen**

```
private void start() throws IOException
{
   ServerSocket serverSocket = new ServerSocket(port);

   System.out.println("Server started at port <" +port +">");
   while(true)
   {
      Socket socket = serverSocket.accept();
      System.out.println("Client connected: <" +socket +">");

      DataOutputStream outStream = new DataOutputStream(socket.getOutputStream());

      connectedSockets.put(socket, outStream);

      ConnectionHandler connectionHandler = new ConnectionHandler(this, socket);
      connectionHandler.start();
      }
   }
}
```

Erbt von Thread und wickelt Verbindung mit dem Client ab

Client/Server Kommunikation in Java VIII

- **Vorsicht bei lesendem/schreibendem Zugriff von mehreren Threads auf Variablen:**



- **Kritische Abschnitte müssen synchronisiert werden. So lange ein lesender/schreibender Zugriff auf eine Variable stattfindet, darf kein anderer Thread darauf zugreifen.**

Client/Server Kommunikation in Java IX

- **Blöcke sychronisieren:**

```
private void start() throws IOException
{
      …
    synchronized (connectedSockets)
    {
       connectedSockets.put(socket, outStream);
    }
      …
}
```

`connectedSockets` ist ein Monitor Objekt über das synchronisiert wird

- **Methoden synchronisieren:**

```
public synchronized void doSomething()
{
      // Do something
}
```

Hier wird über das Objekt selbst synchronisiert

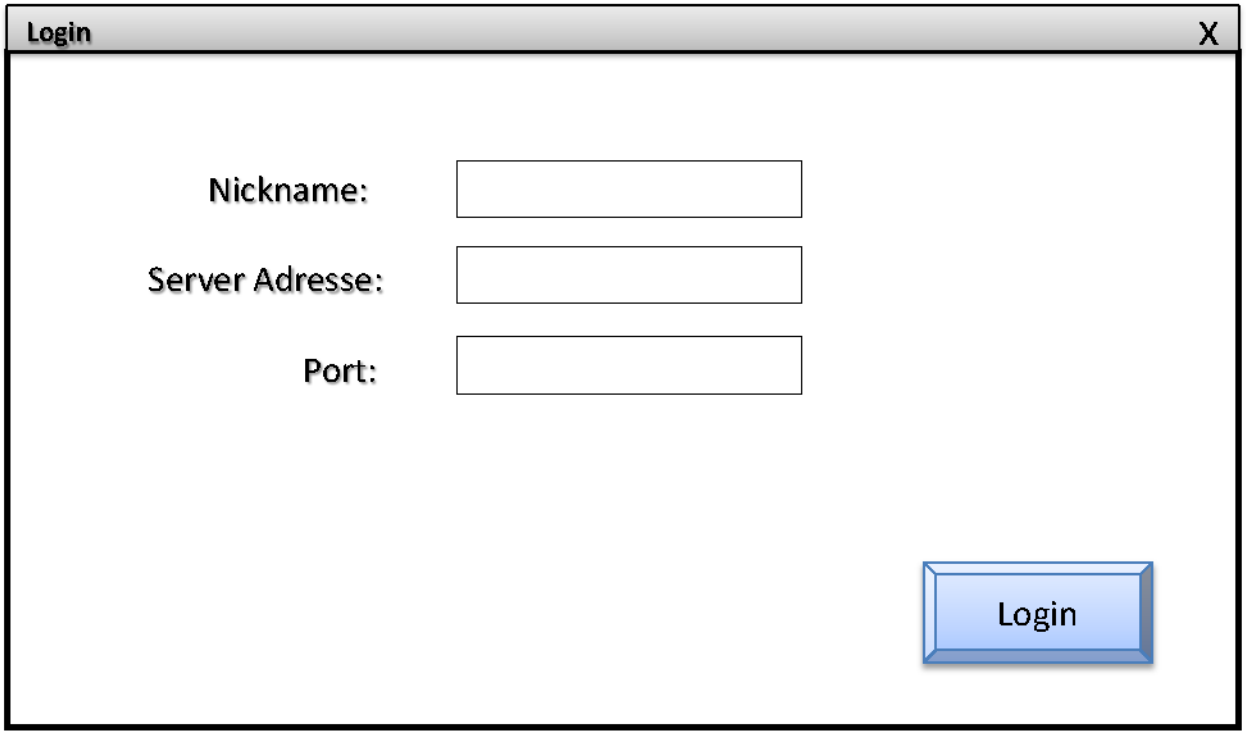
Vorstellung HA11 (Zusatzaufgabe) I

- **HA11 ist eine Zusatzaufgabe, muss also nicht unbedingt bearbeitet werden**
- **Gute Übung im Hinblick auf SE1**

- **Wir wollen einen Chat-Server erstellen**
  - Aufgabe 1: Projekt mit Struktur erstellen
  - Aufgabe 2: Server erstellen
  - Aufgabe 3: Client erstellen
  - Aufgabe 4: Login GUI
  - Aufgabe 5: Chat GUI
  - Aufgabe 6: LoginController
  - Aufgabe 7: ClientController

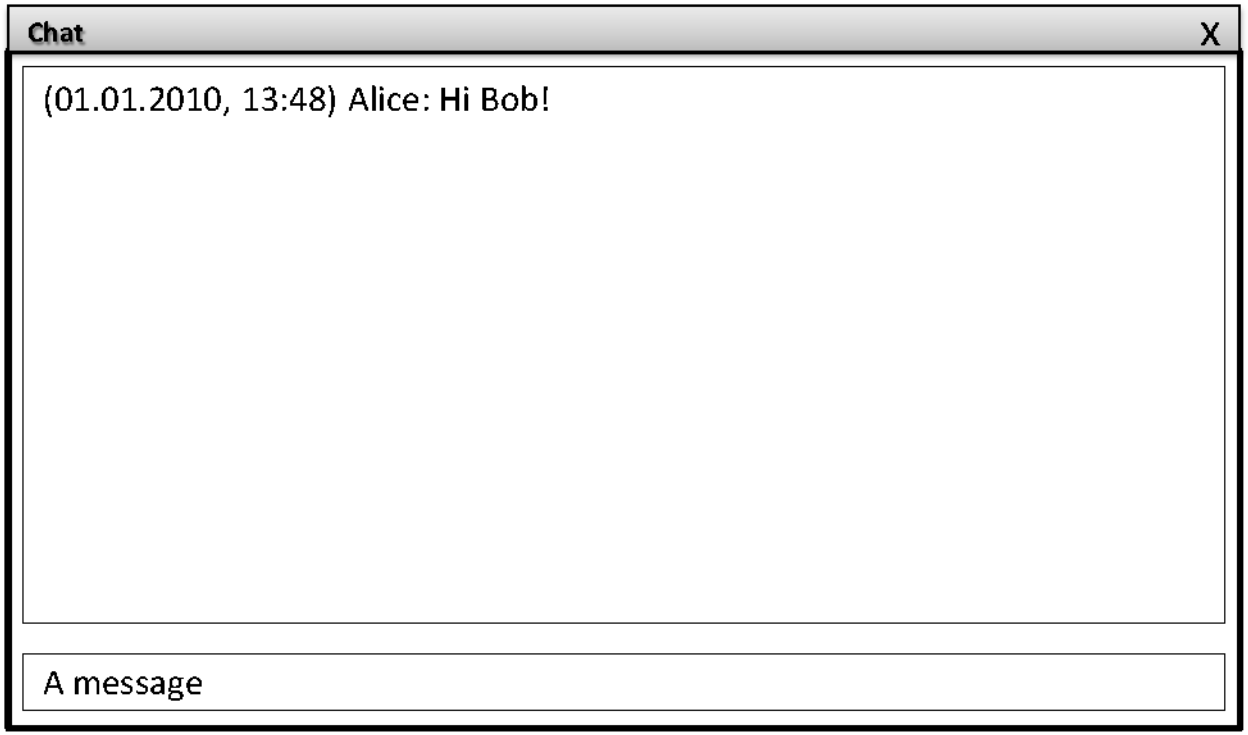
Vorstellung HA11 (Zusatzaufgabe) II

- **Login Mockup:**



Vorstellung HA11 (Zusatzaufgabe) III

- **Chat GUI Mockup:**



Praktische Übung

- **Entwickelt einen Server, der mehrere Verbindungen entgegennehmen kann**
- **Entwickelt einen Client, der sich an diesen Server connecten kann**

Ende

Schönes WE!