Prof. Dr. A. Poetzsch-Heffter
Mathias Weber, M.Sc.

University of Kaiserslautern
Department of Computer Science
Software Technology Group

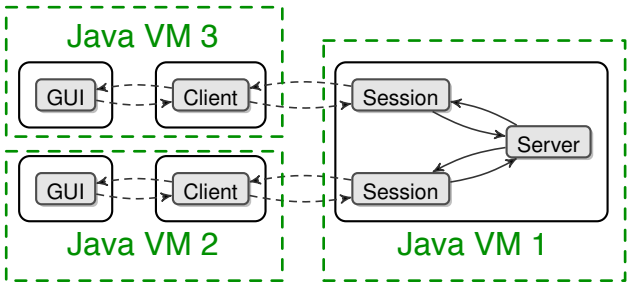
Advanced Aspects of Object-Oriented Programming (SS 2015)

Practice Sheet 12

Date of Issue: 08.07.15
Deadline: 14.07.15
(before the lecture as PDF via E-Mail)

Exercise 1 RMI Chatsystem

In the lecture a chatsystem that runs in a single JavaVM was presented. We want to distribute it according to the following picture.



The communication between the different Java VMs shall be done by RMI. The implementation should be done in plain Java instead of JCoBox.

Implement the complete system. You can use the interfaces definitions presented in the JCoBox slides as a starting point. The GUI should consist of an area the chat is shown in and an input field to enter the message to send. As in the exercises before, the server part of your system shall be able to handle an arbitrary number of clients, which may connect and disconnect at any time, and it shall support the `.bye` and `.history n` commands.

Exercise 2 Mobile Code with RMI

Mobile code gives to possibility to transfer code from one machine to the other and execute it.

a) Download the mobile code project from the lecture homepage and try to get the server to run using the ant command `ant runServer` on a console in the `server` directory.

b) In the `src` folder in the `client` folder you find the needed interfaces `IServer` and `Task`. The RMI registry gets started on the server and an instance of the class `Server` gets registered under the name `Server`. Implement a client that runs the following tasks on the server:

- Read the content of file `secret.txt` in the server directory and print the content to the client console.
- Write a file `output.txt` in the server directory with the content "Hello from Client".
- Delete the file `secret.txt` in the server directory.

Hint: Inform yourself about the `codebase` *property and how to use it to download code to the server. If the client and the server run on the same machine you can use the* `file:` *protocol in the codebase attribute.*

Exercise 3 Swing

As the Swing-Framework is not thread-safe, two rules apply to each swing-application (from the javadoc):

- *Time-consuming tasks should not be run on the Event Dispatch Thread. Otherwise the application becomes unresponsive.*
- *Swing components should be accessed on the Event Dispatch Thread only.*

a) Download the file `worker_JCoBox.zip` from the lecture homepage and unpack it. Implement a CoBox `FactorizeWorker` that performs the actual work of factorizing a given number and notifies the `Controller` about the intermediate results. The CoBox must at least understand the messages **`void`** `execute()` and **`void`** `cancel()` and it must have a constructor that takes a `Controller` and the number of type **`long`** to factorize.

Hint: The annotation `@Swing` on the `Controller` CoBox has the result that all asynchronous calls on the `Controller` box are executed in the Swing Event Dispatch Thread.

b) `SwingWorker` is designed for situations where you need to have a long running task run in a background thread and provide updates to the UI either when done, or while processing.

Rewrite the program in Java and use a `SwingWorker` to perform the factorization.

c) Which threads are involved in the execution of a `SwingWorker`? What is the interface between the GUI and the worker?