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 10 (Hints and Comments)

Exercise 1 Multi-threaded Chatsystem

See provided sources. The implementation is one possibility to implement a chat server, which accepts clients and does not block the chat if a lengthy operation is executed for one client (part b). However, it does not handle the possible exceptions in a nice way, because exceptions may lead to arbitrary behavior. Note, that both implementations guarantee that all clients see the messages in the same order, because all writing to the stream (or to the queue) is ordered by the synchronization on the sessions-object.

Exercise 2 Synchronization and Shared Objects

The only shared variable is the field m in class D. It is shared between threads t3 and t4. The implementation of D is problematic because it depends on the scheduler which of the locally created List of which thread remains local and which gets shared, this may be unwanted, but nevertheless the implementations are thread-safe. All accesses to the lists are correctly synchronized.

C can be completely unsynchronized. The local list can never be accessed by more than one thread and even if generateObject is called concurrently this does not cause any problems.

The local list of D may be accessed by more than one thread, the synchronization in run is needed. The `generateObject` method has to be synchronized, otherwise the field m may be modified by the other thread between the test and the set. The synchronization on `this.m` is needed because it may be the shared list and can be modified on by the other threads run method. Because all accesses to the lists are guarded by a corresponding synchronized block, the lists itself does not have to be created with the synchronization wrapper. All other synchronization cannot be removed.

It is possible to achieve thread-safety in this example by synchronizing on different objects and at different places. So, the given version is not the only one.

Exercise 3 Asynchronous vs. synchronous Calls

a) Synchronous calls: all methods are executed in the order they appear in the snippet. Output:

```
i=5
i=9
```

Asynchronous calls: it is unknown, when the methods are executed:

```
i=5
i=9
```

or, if C handles the message send by B before the one of A

```
i=9
i=5
```

b) With synchronous communication methods are executed completely before the control returns to the caller. That means that the order of execution of methods corresponds to the order of the send messages. In asynchronous contexts, the order of messages may change. Depending on the language, certain guarantees about the message order can be given, but in most cases the order will not be total but some partial order. E.g. in JCoBox it is guaranteed, that if an object a of CoBox a sends two messages to objects in another CoBox, these messages will be handled in the same order as they have been send, but the system makes no guarantees about messages send to or by other CoBoxes.

In multi-threaded contexts the usage of synchronous communication is easier because the behaviour is “easier” to predict. But synchronous communication is a source for deadlocks, which are less likely in asynchronous settings, but for the price of even less predictable behaviour.