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 2013)

Practice Sheet 10 (Hints and Comments)

Exercise 1 The Java Collections Framework and Thread Safety

a) An object (a component, a class,...) is thread-safe, if it behaves according to its specification when executed in a multi-threaded program context. A different possible definition is: An implementation is thread-safe if it is guaranteed to be free of race conditions when accessed by multiple threads simultaneously.

b) They synchronize on the wrapper object and delegate the execution of the calls to the wrapped object. This guarantees, that if all accesses to the wrapped collection are done via the wrapper, the collection is thread-safe. If wrapped the object is directly accessed, the behavior is unspecified.

c) The thread executing the code may get interrupted after the call to `containsKey`, another thread can now modify m and when the first thread continues, his assumption about the state of m is false. Solution: synchronize on m, for the parts where you need exclusive access to m.

```
Map<String,String> m = Collections.synchronizedMap(new HashMap<String,String>(...));
synchronized(m) {
   m.put("a", "b");
   if !(m.containsKey("a")) {
         m.put("c", "d");
   }
}
```

d) The ConcurrentHashMap allows simultanous reads from the map, reading may also overlap writing, but it is ensured, that a read returns only results from writes that have been completed. The wrapper serializes all access to the underlying map. The `ConcurrentHashMap` needs less external synchronization in many cases, but in the same time, the results of read operations get even more unpredictable, additionaly the behavior of iterators over the map changed.

e) Same problem as above. But now multiple thread can be concurrently in the Map implementation, in order to guarantee mutual exclusive access for a thread, we have to protect all possible concurrent accesses with a synchronize block.

Exercise 2 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 3 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 are 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 do 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.