

Advanced Aspects of Object-Oriented Programming (SS 2010)

Practice Sheet 10 (Hints and Comments)

Exercise 1 The Java Collections Framework and Thread Safety

- An object (a component, a class,...) is thread-safe, if it behaves according to its specification when executed in a multi-threaded program context.
- 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 object is directly accessed, the behavior is unspecified.
- 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");
    }
}
```

- 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, additionally the behavior of iterators over the map changed.
- Same problem and same fix as above.

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 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. So no synchronization can be removed.