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

Date of Issue: 18.06.13
Deadline: 25.06.13
(before the lecture as PDF via E-Mail)

Exercise 1 The Java Collections Framework and Thread Safety

a) Explain the term "thread-safe".

b) Inform yourself about the synchronization wrappers for the collection classes of `java.util`. How do they work and which guarantees are given?

c) Are there problems with the following code, if it is executed in a multi-threaded program context? If yes, explain and fix the issues.

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

d) Java 5 introduced the package `java.util.concurrent`, that contains classes to support common tasks in multi-threaded programs. Consider the class `ConcurrentHashMap`. What is the difference between a `ConcurrentHashMap` and a synchronized Map as returned by the synchronization wrapper? Which are the (dis-)advantages of using a `ConcurrentHashMap` over a synchronized Map, and vice versa?

e) Are there problems with the following code, if it is executed in a multi-threaded program context? If yes, explain and fix the issues.

```
ConcurrentHashMap<String,String> m = new ConcurrentHashMap<String,String>(...);
m.put("a", "b");
if !(m.containsKey("a")) {
    m.put("c", "d");
}
```

Exercise 2 Multi-threaded Chatsystem

In this exercise we want to develop a simple Chatserver, that can be used with telnet as client. You can use the code provided on the webpage as basis for your implementation. The provided server accepts a single client, receives its input and closes the connection after the client sends ".bye" and terminates itself.

The following shows you a sample session. The server side:

```
> java ChatServer 12345
Binding to port 12345, please wait  ...
Server started: ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=12345]
Waiting for a client ...
Client accepted: Socket[addr=/0:0:0:0:0:0:0:1,port=50540,localport=12345]
Received: Sending some messages
Received: Some more
Received: and quit now
Received: .bye
>
```

Client side:

```
> telnet localhost 12345
Trying ::1...
Connected to localhost.
Escape character is '^]'.
Server: Connected to chatserver
Sending some messages
Some more
and quit now
.bye
Connection closed by foreign host.
>
```

a) Modify server and client, such that the server can handle multiple clients at the same time. All messages shall be displayed immediately at all clients. All messages shall be prefixed with a unique id for each client.

Hints: Create a thread for each client and let this thread handle the communication. The main server thread is then available to accept new incoming clients.

b) To allow clients to recall the discussion, implement the command ".history n", with n being an integer. This command sends the last n messages of the chat to the requesting client.

Consider the following questions. If needed change your implementation such that these questions are negated.

- Does sending a long history influence the chat for other clients? Can it block the server or cause lags?
- Is it possible that the client, which requested the history, misses messages by others while receiving the history?

Exercise 3 Synchronization and Shared Objects

Consider the following code. Which variables are shared between the threads? Is this code thread-safe? Remove as much synchronization actions as possible.

```
class C implements Runnable {
    void run() {
        Object[] a = {"a", "b"};
        List<Object> m = Collections.synchronizedList(new LinkedList<Object>(Arrays.asList(a)));
        while (true) {
            synchronized(m) {
                Object o =  generateObject(m);
                if !(m.contains(o)) {
                    m.add(o);
            }
        }
    }

    synchronized Object generateObject (List m) {
        return new Object();
    }
}

class D implements Runnable {
    private List m;

    void run() {
        Object[] a = {"a", "b"};
        List<Object> m =  Collections.synchronizedList(new LinkedList<Object>(Arrays.asList(a)));
        while (true) {
            synchronized(m) {
                Object o =  generateObject(m);
                if !(m.contains(o)) {
                m.add(o);
            }
        }
    }

    synchronized Object generateObject (List m) {
        if (this.m == null) this.m = m;
        synchronized (this.m) {
            Object o = .... // calculate o, based on the parameter m
            if (!this.m.contains(o)) {
                this.m.add(o);
            }
        }
        return o;
    }
}

public static void main(String ... a) {
    C c = new C();
    Thread t1 = new Thread(c);
    Thread t2 = new Thread(c);
    t1.start();
    t2.start();
    D d = new D();
    Thread t3 = new Thread(d);
    Thread t4 = new Thread(d);
    t3.start();
    t4.start();
}
```