CONCURRENCY
Exercise 09

Marcel ZAUDER 16-124-836
Pascal GERIG 16-104-721

9.1 Several Questions

a) *What criteria might you use to prioritize threads (list at least 5 different criteria)?*
   (i) Based on the due time/desired completion time of a process.
   (ii) Based on any representation of priorization, i.e. task priority, cost etc.
   (iii) Based on the time the service was requested (FIFO).
   (iv) Based on the expected time a thread needs to be served.
   (v) Based on how many other tasks are waiting for the result of this particular task.

b) *What are different possible definitions of fairness (list at least 3 different definitions)?*
   (i) *Weak Fairness*
   A transition or a process should not wait an unbounded amount of time to execute if it is enabled continuously.
   (ii) *Strong Fairness*
   A transition or a process should not wait an unbounded amount of time to execute if it is enabled infinitely often.
   (iii) *Linear Waiting*
   A transition or a process making a request will be served before any other process was served more than once.
   (iv) *FIFO*
   A transition or a process making a request will be served before any other process that made a request after that.

c) *What are Pass-Throughs?*
With *Pass-Throughs* the host maintains a set of immutable references to helper objects. All messages are then relayed to these within unsynchronized methods.

d) *What is Lock-Splitting?*
In *Lock-Splitting* instead of using the same lock for each method of a class an individual lock is created for each method, i.e. one lock for writing and one lock for reading.

e) *When should you consider using optimistic methods (list at least 3 different enablers)?*
   (i) Clients can tolerate either failure or retries.
   (ii) Avoidance or Coping of livelocks.
   (iii) Before a failure occurs the program can rollback into a non-fail state and try again.
   (iv) The chance of a collition is negligible.

Marcel ZAUDER 16-124-836
Pascal GERIG 16-104-721

9.3 Additional Several Questions

a) *How do threads waiting in a Thread.join() loop get aware of that thread's termination?*
When *t.join()* is called the thread/process which is calling that method is waiting while the referenced thread is ALIVE. After *t* terminates the *join()* will also leave the wait-loop and terminate. It is also possible to wait for a given amount of time and if the referenced thread throws an *InterruptedException* the *join*-method will as well.

b) *How could you optimize the code below?*

```
Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
                <insert your code here>
        }
});
t.start();
t.join();
```

Because the thread is started and directly afterwards it is joined, there is no need for a thread implementation, but the code in the run method itself can just be executed directly.

c) *Are String objects in Java mutable or immutable? Justify your answer!*
A STRING object in Java is immutable, because once it is generated this particular instance cannot be changed. In order to create strings on runtime, A StringBuilder should be used. Also there does not exist any method call on a STRING object that changes the object itself but creates a new modified STRING object.

d) *Does the FSP progress property below enforce fairness? Justify your answer!*

```
progress HeadsOrTales = {head, tale}
```

No, it does not, because it only ensures that either the action *head* or *tale* can be executed at any time, but it does not ensure that it is actually executed.