SolutionSerie 4 - Safety Patterns

Exercise 1

Answer the following questions: (0.5 points each)

1. Why are immutable classes inherently safe? **Answer:**

   Because the state of an instance of an immutable class cannot be changed after its creation. This prevents the object from ever getting into an inconsistent state. To create an immutable class you must ensure that the instance fields are never changed after the construction e.g. by declaring them final.

2. What is "balking"? **Answer:**

   Balking describes a design pattern that is used in conjunction with state-dependent actions. In practice, balking consists in returning a failure (or rather a refusal to do something) if some pre-conditions fail, i.e. some object fails to satisfy the class invariant. A requesting object is then informed, e.g. by raising an IllegalStateException.

3. When is partial synchronization better than full synchronization? **Answer:**

   Partial synchronization should be preferred over full synchronization when (1) objects encapsulate both mutable and immutable data, i.e contain changeable and unchangeable instance variables, (2) methods can organized such that parts of them deal with critical sections and other parts do not. Then only the "critical section"-parts need to be synchronized. This reduces synchronization overhead.

4. How does containment avoid the need for synchronization? **Answer:**

   By embedding unsynchronized objects inside other objects that have at most one active thread at a time. The "contained" objects can be regarded as exclusively held resources. Since such objects do not reveal their identity to other objects (i.e. not accessible via public methods etc.), they do not need to be synchronized.

5. What liveness problems can full synchronization introduce? **Answer:**

   For example when one method contains two separate critical sections then a potential deadlock may occur. Or nested monitor problem: if not properly implemented, for instance two levels of synchronization lock may lead to deadlock (using full synchronization) when a thread releases the lock for one resource but not for the second resource. Another thread might wait forever for the second resource to be unlocked..

6. When is it all right to declare only some methods as synchronized? **Answer:**

   For example if some method accesses only immutable data it does not have to be declared as synchronized. See partial synchronization.

Exercise 2 (2 points)

The dinning savages: A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he helps himself from the pot unless it is empty in which case he waits for the pot to be filled. If the pot is empty the cook refills the pot with M servings. The behavior of the savages and the cook are described by:

```
SAVAGE = (getserving -> SAVAGE).
COOK   = (fillpot -> COOK).
```

Model the behavior of the pot as an FSP process. **Answer:**

```
const M = 5
SAVAGE = (getserving -> SAVAGE).
COOK   = (fillpot -> COOK).
POT              = SERVINGS[0],
SERVINGS[i:0..M] = (when (i==0) fillpot     -> SERVINGS[M]
                   |when (i>0 )  getserving -> SERVINGS[i-1]
                   ).

||SAVAGES = (SAVAGE || COOK || POT).
```

Exercise 3 (2 points)

What action trace violates the following safety property?

```
property PS = (a -> (b -> PS|a -> PS)|b -> a -> PS).
```

Answer:

Trace to property violation in PS:
b
b

Exercise 4 (3 points)