2003 Paper 5 Question 4

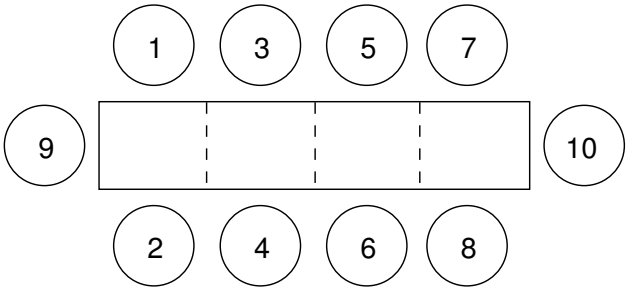
Concurrent Systems and Applications

In a concurrent system it is usual to distinguish between *safety properties* and *liveness properties*.

(*a*) What is meant by *mutual exclusion*, and how does a programmer ensure it is enforced in Java? Indicate whether it relates to safety or to liveness. [4 marks]

(*b*) What is *deadlock* and how can it occur? Again, indicate whether it relates to safety or to liveness. [4 marks]

A group of ten *feeding philosophers* has gathered to eat at a long trough, as shown below. They eat by leaning over the trough and so must be careful not to let their heads collide in the middle. To ensure safety they decide that around each position in the trough at most one philosopher is allowed to be eating at any one time. For instance, only one of 1, 2 or 9 could be eating at once, but there is no problem with 9, 3, 6 and 10 all eating together.



In a Java system, each philosopher is modelled by a thread which loops between eating and non-eating phases. There are four `TroughPosition` objects, corresponding to the four positions along the trough. Each philosopher has a reference to the position at which he or she is eating.

(*c*) Define a Java class `TroughPosition` which supports methods `startEating()` and `finishEating()` to ensure safe execution for the philosophers at that position. [6 marks]

(*d*) Can your solution suffer from deadlock? Either explain why it cannot occur, or explain how it could be avoided. [3 marks]

(*e*) Can your solution suffer from starvation – that is, can one thread continuously remain in the `startEating()` method? Either explain why this cannot occur, or explain how it could be avoided. [3 marks]