assignment

1.

Implement a synchronization mechanism similar to the mechanism provided by java within the **`java.util.concurrent packages`** (Explicit Locks and Condition variables) but whose behaviour is in accordance with the semantic "*signal-and-urgent*".
For the implementation of this mechanism you can use only the built-in synchronization constructs provided by Java (i.e. **`synchronized`** blocks or **`synchronized`** methods) and the methods **`wait()`**, **`notify()`** and **`notifyAll()`** provided by the class **`Object`**).
In particolar:

- implement the class **`FairLock`** that provides the two methods **`lock()`** and **`unlock()`**, to be used to explicitly guarantee the mutual exclusion of critical sections.
  Your implementation must guarantee that threads waiting to acquire a **`FairLock`** are awakened in a FIFO order.
- implement also the class **`Condition`** that provides the two methods **`await()`** and **`signal()`** that are used, respectively, to block a thread on a **`Condition`** variable and to awake the first thread (if any) blocked on the **`Condition`** variable. In other words **`Condition`** variables must be implemented as FIFO queues. The semantics of the **`signal`** operation must be "*signal and urgent*".
  Remember that every instance of the class **`Condition`** must be intrinsically bound to a lock (instance of the class **`FairLock`**). For this reason, the class **`FairLock`** provides, in addition to methods **`lock() and unlock()`**, also the method **`newCondition()`** that returns a new **`Condition`** instance that is bound to this **`FairLock`** instance.

assignment

2.0

As a simple example of the use of the previous mechanism, implement a manager of a single resource that dynamically allocates the resource to three client threads: **`ClientA1`**, **`ClientA2`** and **`ClientB`**.
If the resource is in use by ClientA1 or by ClientA2, when it is released and both ClientB the other ClientA are waiting for the resource, ClientB must be privileged.

2.1

Provide also the implementation of the same manager but now by using the analogous mechanism provided by Java (**`Lock`** and **`Condition`** variables whose behaviour is in accordane with the semantics *signal-and-continue* and point out the differences, if any, between this implementation and the previous one.

assignment

3.0

By using the language **`PSF`**, provide the *design model* of the problem described at point 2.0 .

3.1

From the design model described at point 3.0, derive the corresponding java program implemented by using the **`Lock`** and **`Condition`** variables provided by java and whose behaviour is in accordane with the sematics *signal-and-continue*.

3.2

By modeling this implementation with the *FSP* language, verify that it satisfies the problem’s specification.