The Dining Philosophers Problem

A Monitor-Based Solution
(coded in the KPL language)
The Philosophers’ Threads

function PhilosphizeAndEat (p: int)
    var i: int
    for i = 1 to 7
        -- Now he is thinking
        mon.PickupForks (p)
        -- Now he is eating
        mon.PutDownForks (p)
    endFor
endFunction
Startup Code

```javascript
var
    mon: ForkMonitor
    philosopher: array [5] of Thread =
        new array of Thread { 5 of new Thread }
...

-- Initialize the monitor...
mon = new ForkMonitor
mon.Init ()
mon.PrintAllStatus ()

-- Start up a thread for each philosopher...
philosopher[0].Init ("Plato")
philosopher[0].Fork (PhilosophizeAndEat, 0)
    ...etc...
philosopher[4].Init ("Aristotle")
philosopher[4].Fork (PhilosophizeAndEat, 4)
```
The Monitor

class ForkMonitor
  superclass Object
  fields
    monitorLock: Mutex
    status: array [5] of int
      -- HUNGRY, EATING, or THINKING
    startEating: array [5] of Condition
      -- Signaled when eating can begin
  methods
    Init ()
    PickupForks (p: int)       -- Entry Method
    PutDownForks (p: int)      -- Entry Method
    CheckAboutEating (p: int)  -- Local Method
    PrintAllStatus ()          -- Local Method
endClass
method Init ()
   -- Initialize so that all philosophers are
   -- THINKING. Also create the monitor lock
   -- and the 5 condition variables.
   var i: int
   status = new array of int { 5 of THINKING } 
   startEating = new array of Condition 
      { 5 of new Condition }
   for i = 0 to 4
      startEating[i].Init ()
   endFor
   monitorLock = new Mutex
   monitorLock.Init ()
endMethod
method PickupForks (p: int)
    -- This method is called when philosopher 'p'
    -- wants to eat. Change his status to HUNGRY
    -- and then see if he can begin eating. If he
    -- was not able to begin immediately, then
    -- this thread must wait.
    monitorLock.Lock ()
    status [p] = HUNGRY
    self.PrintAllStatus ()
    self.CheckAboutEating (p)
    if status [p] != EATING
        startEating [p].Wait (& monitorLock)
    endIf
    monitorLock.Unlock ()
endMethod
PutDownForks

method PutDownForks (p: int)
    -- This method is called when the philosopher 'p'
    -- is done eating. Change his status. Also,
    -- this might make it possible for his left and
    -- right neighbors to begin eating, so check
    -- on them.
    monitorLock.Lock ()
    status [p] = THINKING
    self.PrintAllStatus ()
    self.CheckAboutEating ((p+1) % 5)
    self.CheckAboutEating ((p-1) % 5)
    monitorLock.Unlock ()
endMethod
method CheckAboutEating (p: int)
    -- See if the p-th philosopher should begin
    -- eating. He should begin if he is HUNGRY and
    -- if his left and right neighbors are not
    -- eating. If so, change his status to EATING.
    -- Also, it could be that philosopher p's
    -- thread is waiting. Signal that thread's
    -- condition so it can resume, if it is waiting.
    if status [p] == HUNGRY &&
        status [(p+1) % 5] != EATING &&
        status [(p-1) % 5] != EATING
        status [p] = EATING
        self.PrintAllStatus ()
        startEating [p].Signal (& monitorLock)
    endIf
endMethod
PrintAllStatus

method PrintAllStatus ()
    -- This is a "local" method.
    var p: int
    for p = 0 to 4
        switch status [p]
            case HUNGRY:
                print ("    ")
                break
            case EATING:
                print ("E   ")
                break
            case THINKING:
                print (".   ")
                break
        endSwitch
    endFor
    nl ()
endMethod