

4. Exemplary Solutions: LAN Simulation

Exercise 4.1

Trivial.

Exercise 4.2

```
Node >> name: aSymbol  
       name := aSymbol
```

```
Node >> nextNode: aNode  
       nextNode := aNode
```

Exercise 4.3

```
Node >> hasNextNode  
       ^self nextNode notNil
```

Exercise 4.4

```
Node >> printOn: aStream  
       aStream nextPutAll: 'Node named: ', self name asString.  
       self hasNextNode ifTrue: [  
           aStream nextPutAll: ' connected to: ',  
                               self nextNode name asString.  
       ]
```

Exercise 4.5

```
Node >> accept: aPacket  
       "Having received the packet, send it on. This is the default  
       behavior. My subclasses will probably override me to do  
       something special"  
  
       self send: aPacket.
```

```
Node >> send: aPacket  
       "Precondition: self have a nextNode"  
       "Using self assert: self hasNextNode is also possible"  
       self hasNextNode ifTrue: [  
           self nextNode accept: aPacket.  
  
           "Display debug information in the Transcript, then  
           send a packet to my following node"
```

```
Transcript show:
  self name printString,
  ' sends a packet to ',
  self nextNode name printString; cr.
].
```

Exercise 4.6

Trivial.

Exercise 4.7

```
PacketTest >> testPrintOn
  | node1 pc1 packet |

node1 := Node new name: #Node1.
pc1 := Node new name: #PC1.
node1 nextNode: pc1.
packet := Packet new
  originator: node1;
  addressee: pc1;
  contents: 'a message'.

self assert: packet printString =
  'Packet from: Node1 to: PC1 with contents: a message'.

Packet >> printOn: aStream
  aStream nextPutAll: 'Packet from: ', self originator name asString.
  aStream nextPutAll: ' to: ', self addressee name asString.
  aStream nextPutAll: ' with contents: ', self contents.
```

Exercise 4.8

Trivial.

Exercise 4.9

```
Workstation >> accept: aPacket
  aPacket addressee = self ifTrue: [
    Transcript show: 'A packet is accepted by the Workstation ',
    self name asString.
  ] ifFalse: [super accept: aPacket]
```

Exercise 4.10

```
Node >> originate: aPacket
    "This is how packets get inserted into the network.
    This is a likely method to be rewritten to permit
    packets to be entered in various ways. Currently,
    I assume that someone else creates the packet and
    passes it to me as an argument."

    aPacket originator: self.
    self send: aPacket.
```

Exercise 4.11

```
Node subclass: #Printer
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'LAN'

Printer >> accept: aPacket
    aPacket addressee = self ifTrue: [
        Transcript show: aPacket.
    ] ifFalse: [super accept: aPacket]
```

Exercise 4.12

```
Node subclass: #Logger
    instanceVariableNames: 'forwardedPackets addressedToMePackets'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'LAN'

Logger >> initialize
    forwardedPackets := OrderedCollection new.
    addressedToMePackets := OrderedCollection new.

Logger >> forwardedPackets
    ^ forwardedPackets

Logger >> addressedToMePackets
    ^ addressedToMePackets
```

```
Logger >> forwardedPacket: aPacket
    self forwardedPackets add: aPacket.

Logger >> addressedToMePacket: aPacket
    self addressedToMePackets add: aPacket.

Logger >> accept: aPacket
    aPacket addressee = self name ifTrue: [
        self addressedToMePacket: aPacket.
    ] ifFalse: [
        self forwardedPacket: aPacket.
        super accept: aPacket.
    ]
```

Exercise 4.13

```
TestCase subclass: #LANTest
    instanceVariableNames: 'mac node1 node2 node3 pc logger printer'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'LAN'
```

```
LANTest >> setUp
    mac := Workstation new name: #Mac.
    node1 := Node new name: #Node1.
    node2 := Node new name: #Node2.
    node3 := Node new name: #Node3.
    pc := Workstation new name: #PC.
    logger := Logger new name: #Logger.
    printer := Printer new name: #Printer.
```

```
mac nextNode: node1.
node1 nextNode: node2.
node2 nextNode: logger.
logger nextNode: node3.
node3 nextNode: pc.
pc nextNode: printer.
printer nextNode: mac.
```

```
LANTest >> testPacketMacToPrinter
    packet := Packet new
        originator: mac;
        addressee: printer;
        contents: 'message to be printed'.
```

```
mac originate: packet.

self assert: (logger forwardedPackets includes: packet).

LANTest >> testPacketMacToLogger
| packet |
packet := Packet new
    originator: mac;
    addressee: logger;
    contents: 'message for the logger'.

mac originate: packet.

self assert: (logger addressedToMePackets includes: packet).
self deny: (logger forwardedPackets includes: packet).
```

Exercise 4.14

```
Node >> accept: aPacket
    "checking whether the packet's originator is the current node"
    aPacket originator ~= self ifFalse: [
        self send: aPacket.
    ].

LANTest >> testLoopSolution1
| packet unknown |
unknown := Node new name: #Unknown.
packet := Packet new
    originator: mac;
    addressee: unknown;
    contents: 'message without valid node'.

mac originate: packet.
self assert: (logger forwardedPackets includes: packet).
```

Exercise 4.15

```
Packet >> initialize
    "The packet keeps tracks of all the nodes it passed by in a collection"
    visitedNodes := OrderedCollection new

Packet >> passedBy: aNode
    visitedNodes add: aNode
```

```
Packet >> hasPassedBy: aNode
  ^ visitedNodes includes: aNode

Node >> accept: aPacket
  "checking whether this packet already passed by a node"
  (aPacket hasPassedBy: self) ifFalse: [
    aPacket passedBy: self.
    self send: aPacket.
  ].

LANTest >> testLoopSolution2
  | packet unknown |
  unknown := Node new name: #Unknown.
  packet := Packet new
    originator: mac;
    addressee: unknown;
    contents: 'message without valid node'.

  mac originate: packet.
  self assert: (logger forwardedPackets includes: packet).
  self assert: (packet hasPassedBy: node1).
  self assert: (packet hasPassedBy: node2).
  self assert: (packet hasPassedBy: node3).
  self assert: (packet hasPassedBy: pc).
  self assert: (packet hasPassedBy: logger).
  self assert: (packet hasPassedBy: printer).
```
