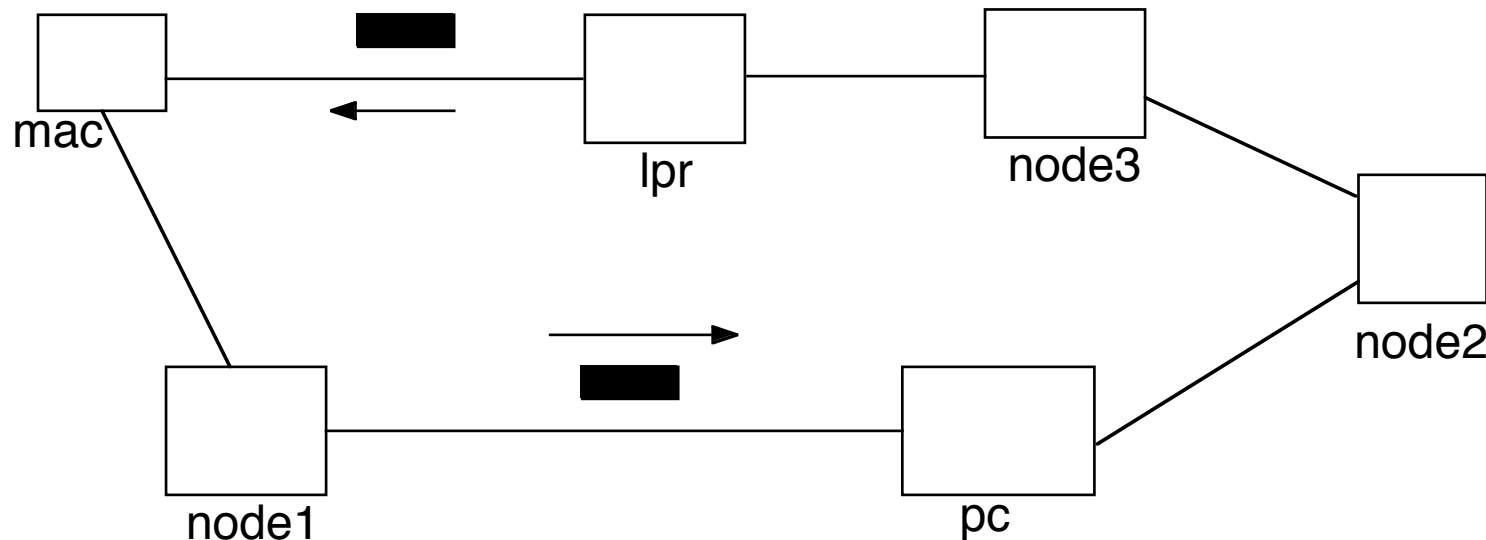


Let's Play Objects

- Simulate a LAN physically
- Set up a context for
 - future chapters
 - Exercises
- Some forward references to intrigue you

A LAN Simulator

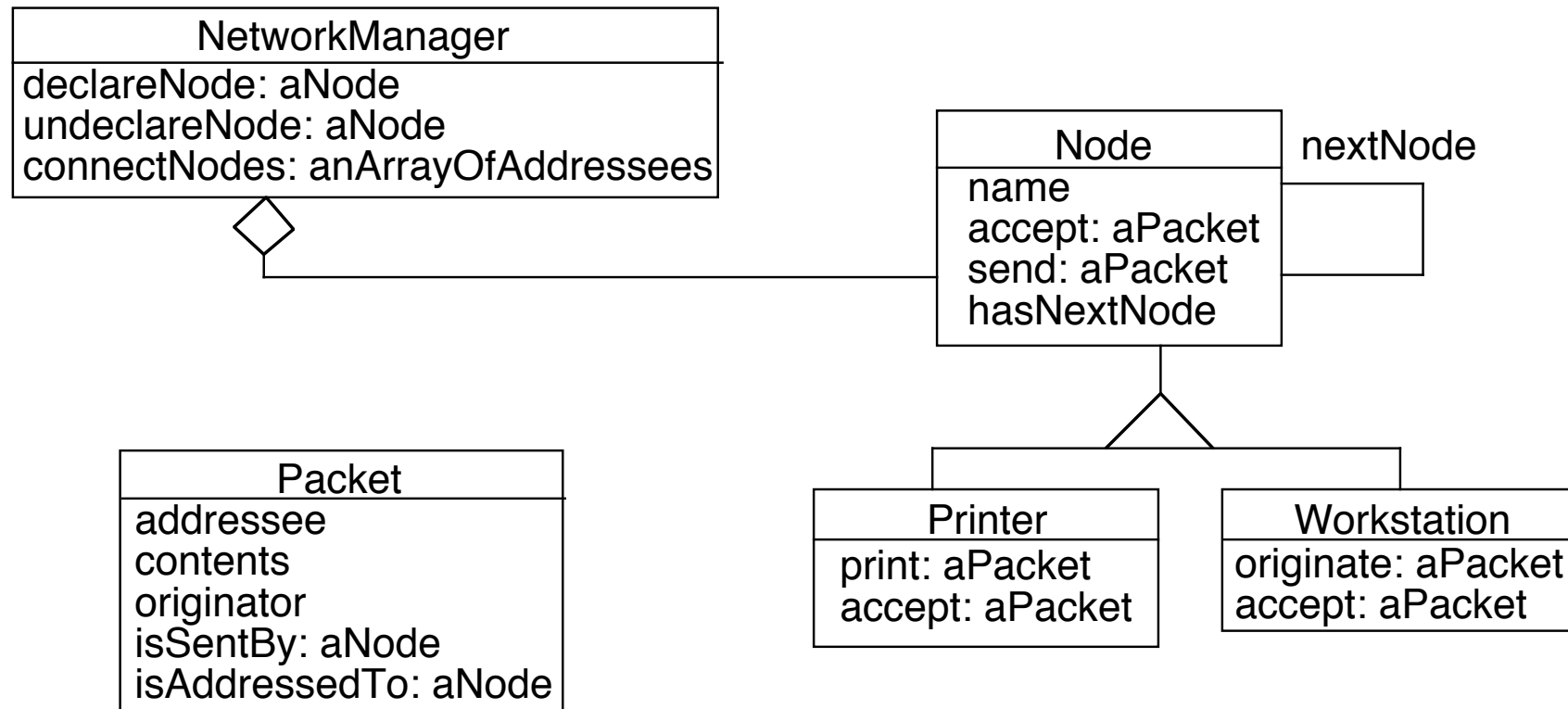
- A LAN contains nodes, workstations, printers, file servers. Packets are sent in a LAN and each node treats them differently.



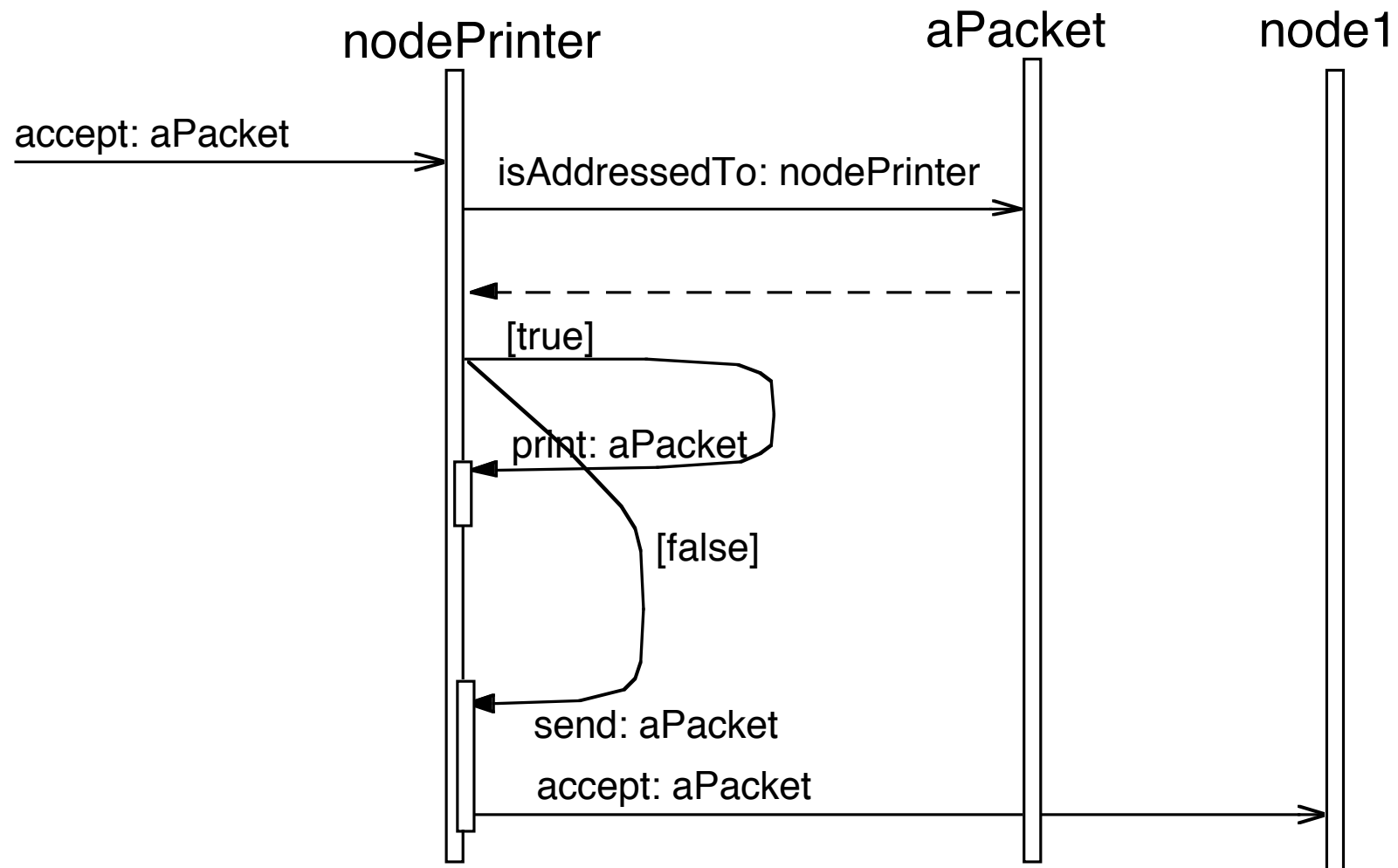
Three Kinds of Objects

- Node and its subclasses represent the entities that are connected to form a LAN.
- Packet represents the information that flows between Nodes.
- NetworkManager manages how the nodes are connected

LAN Design



Interactions Between Nodes



Node and Packet Creation

```
|macNode pcNode node1 printerNode node2 node3 packet|
macNode := Workstation withName: #mac.
pcNode := Workstation withName: #pc.
node1 := Node withName: #node1.
node2 := Node withName: #node2.
node3 := Node withName: #node2.
printerNode := Printer withName: #lpr.
macNode nextNode: node1.
node1 nextNode: pcNode.
pcNode nextNode: node2.
node3 nextNode: printerNode.
lpr nextNode: macNode.

packet := Packet send: 'This packet travelled to' to: #lpr.
```

Objects communicate via Messages (II)

- Message: 1 + 2
 - receiver : 1 (an instance of SmallInteger)
 - selector: #+
 - arguments: 2
- Message: lpr nextNode: macNode
 - receiver lpr (an instance of LanPrinter)
 - selector: #nextNode:
 - arguments: macNode (an instance of Workstation)
- Message: Packet send: 'This packet travelled to' to: #lpr
 - receiver: Packet (a class)
 - selector: #send:to:
 - arguments: 'This packet travelled to' and #lpr

The Definition of a LAN

- To simplify the creation and the manipulation of a LAN:

```
| aLan |
```

```
aLan := NetworkManager new.
```

```
aLan createAndDeclareNodesFromAddresses: #(node1 node2 node3)  
ofKind: Node.
```

```
aLan createAndDeclareNodesFromAddresses: #(mac pc) ofKind:  
Workstation.
```

```
aLan createAndDeclareNodesFromAddresses: #(lpr) ofKind:  
LanPrinter.
```

```
aLan connectNodesFromAddresses: #(mac node1 pc node2 node3  
lpr)
```

- Now we can query the LAN to get some nodes:

```
aLan findNodeWithAddress: #mac
```


Transmitting a Packet

| aLan packet macNode|

...

macNode := aLan findNodeWithAddress: #mac.

packet := Packet send: 'This packet travelled to the printer' to: #lpr.

macNode originate: packet.

- > mac sends a packet to pc
- > pc sends a packet to node1
- > node1 sends a packet to node2
- > node2 sends a packet to node3
- > node3 sends a packet to lpr
- > lpr is printing
- > this packet travelled to lpr

How to Define a Class

Fill the template:

```
NameOfSuperclass subclass: #NameOfClass
instanceVariableNames: 'instVarName1'
classVariableNames: 'ClassVarName1 ClassVarName2'
poolDictionaries: ''
category: 'LAN'
```

For example to create the class Packet

```
Object subclass: #Packet
instanceVariableNames: 'addressee originator contents '
classVariableNames: ''
poolDictionaries: ''
category: 'LAN'
```

How to define a method?

message selector and argument names
"comment stating purpose of message"
| temporary variable names |
statements

```
LanPrinter>>accept: thePacket  
"If the packet is addressed to me, print it.  
Otherwise just behave like a normal node."  
(thePacket isAddressedTo: self)  
  ifTrue: [self print: thePacket]  
  ifFalse: [super accept: thePacket]
```

In Java

In Java we would write

```
void accept(thePacket Packet)
/*If the packet is addressed to me, print it.
Otherwise just behave like a normal node.*/
if (thePacket.isAddressedTo(this)){
    this.print(thePacket)}
else super.accept(thePacket)}
```