21.2 Self-Referential Classes

A **self-referential class** contains an instance variable that refers to another object of the same class type. For example, the generic `Node` class declaration

```
class Node<E> {
   private E data;
   private Node<E> nextNode; // reference to next lin
   public Node(E data) { /* constructor body */ }
   public void setData(E data) { /* method body */ }
   public E getData() { /* method body */ }
   public void setNext(Node<E> next) { /* method body
   public Node<E> getNext() { /* method body */ }
}
```

has two `private` instance variables—`data` (of the generic type `E`) and `Node<E>` variable `nextNode`. Variable `nextNode` references a `Node<E>` object, an object of the same class being declared here—hence the term "self-referential class." Field `nextNode` is a **link**—it "links" an object of type `Node<E>` to another object of the same type. Type `Node<E>` also has five methods: a constructor that receives a value to initialize `data`, a `setData` method to set the value of `data`, a `getData` method to return the value of `data`, a `setNext` method to set the value of `nextNode` and

a `getNext` method to return a reference to the next node.

Programs can link self-referential objects together to form such useful data structures as lists, queues, stacks and trees. Figure 21.1 illustrates two self-referential objects linked together to form a list—`15` and `10` are the `data` values in `Node<Integer>` objects. A backslash (\)—representing a `null` reference—is placed in the link member of the second self-referential object to indicate that the link does *not* refer to another object. The backslash is illustrative; it does not correspond to the backslash character in Java. By convention, in code we use `null` to indicate the end of a data structure.

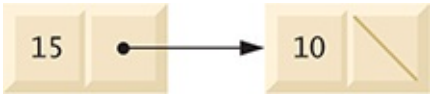

Fig. 21.1

Self-referential-class objects linked together.