CLASS-BASED LANGUAGES

Class-based languages form the mainstream of object-oriented programming. Among the best-known class-based languages are Simula, the first object-oriented language, Smalltalk, the first dynamically typed object-oriented language, and C++, whose main class-based features are modeled after those of Simula. (Bibliographic references to these and other languages can be found in the List of Languages at the back of the book.)

In this chapter we discuss a kernel of properties that, with minimal controversy, characterize classical class-based languages above and beyond their countless peculiarities. More advanced class-based features are treated in Chapter 3.

2.1 Classes and Objects

Class-based languages are centered around the notion of classes as descriptions of objects. We begin with an example of a class declaration.

```plaintext
class cell is
    var contents: Integer := 0;
    method get(): Integer is
        return self.contents;
    end;
    method set(n: Integer) is
        self.contents := n;
    end;
end;
```

A class is intended to describe the structure of all the objects generated from the class. The class cell describes storage-cell objects having an integer field named contents, which is initialized to zero, and two methods named get and set, whose code operates on the contents field.¹ The get method has no parameters; when invoked, it fetches the contents field and returns it. The set method has an integer parameter; it stores the

¹ There is a wide spectrum of class notions in programming languages, from classes detailing values for all fields and code for all methods, to "abstract classes" with as yet uncoded methods, to classes with no specified values or code at all. We take classes to have fully specified values and code. The case where there are no values or code is better covered by object types, described in a later section.
parameter in the contents field and returns nothing. Within these methods, the special identifier self refers to the host object. The fields and methods of an object are collectively called its attributes.

The intended behavior of objects can be understood in terms of the naive storage model of Figure 2-1. In that model, an object is internally represented as a reference to a record of attributes. An operation on an object attribute implicitly bypasses the reference in order to access the attribute. Program variables hold references, not attribute records. Parameter passing and assignment copy references, not the associated attribute records. Therefore parameter passing and assignment produce sharing of attribute records.2

* Figure 2-1. Naive storage model.

An object can be created from a class c by the construction new c. More precisely, new c allocates an attribute record and returns a reference to it. The attribute record contains the initial values and the method code specified by c. Distinct executions of new c produce distinct objects; that is, they produce references to distinct attribute records. An object generated from class c via new is colloquially called “an object of class c” or “an instance of class c”.

We indicate by InstanceTypeOf(c) the type of objects of class c. In the following example, a new cell is bound to the variable myCell with type InstanceTypeOf(cell):

```ml
var myCell: InstanceTypeOf(cell) := new cell;
```

By introducing the type InstanceTypeOf(cell), we have made an important distinction between classes and types. We could have chosen to consider cell, instead of InstanceTypeOf(cell), as the type of the objects generated from cell. However, such an identification of classes with types would cause confusion later.

Given a cell named myCell, we can extract the value of the contents field by myCell.contents, and we can update it by myCell.contents:=n.3 The contents fields of distinct objects of class cell change independently upon updates. That is, each object of class cell has a separate contents field.

---

2 Some languages, such as C++ and Oberon, expose the distinction between attribute records and references to attribute records in order to distinguish between stack-allocated and heap-allocated objects. Other languages, such as Simula, Smalltalk, and Modula-3, hide this distinction and deal exclusively with heap-allocated objects.