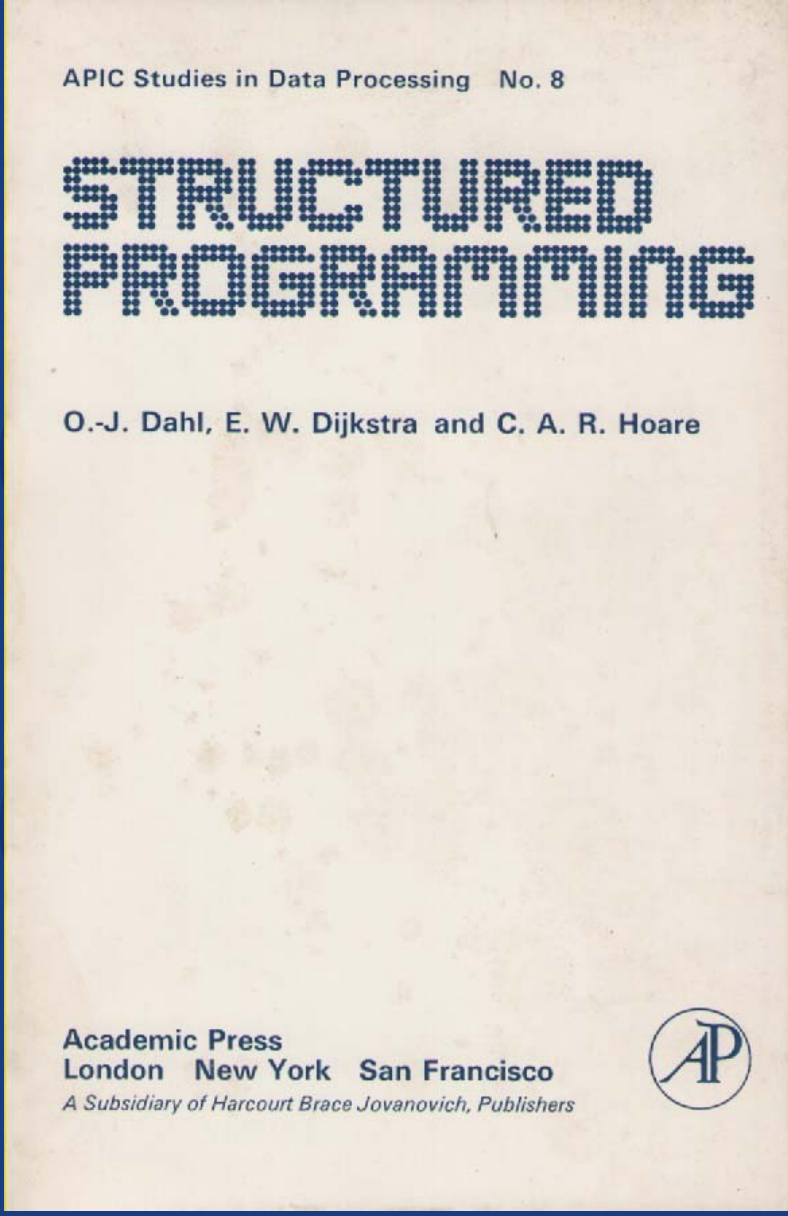



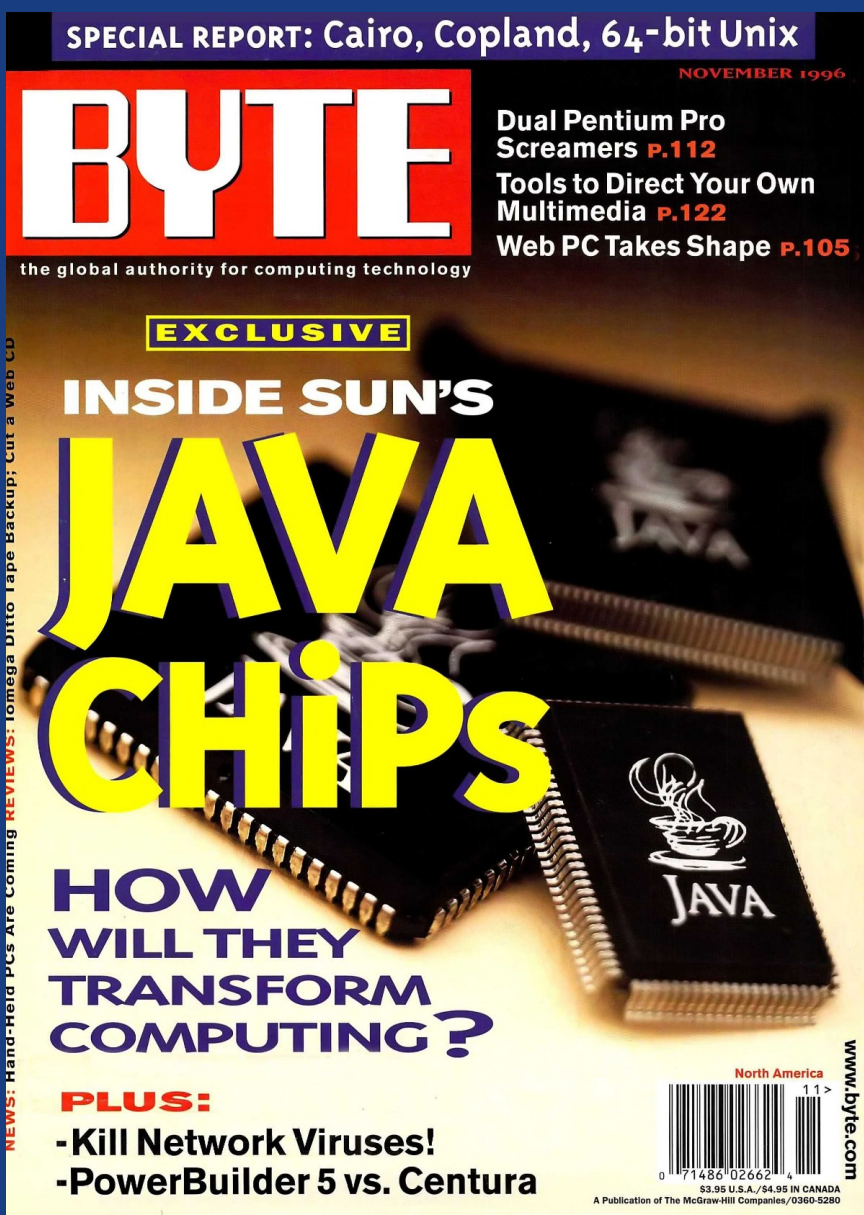
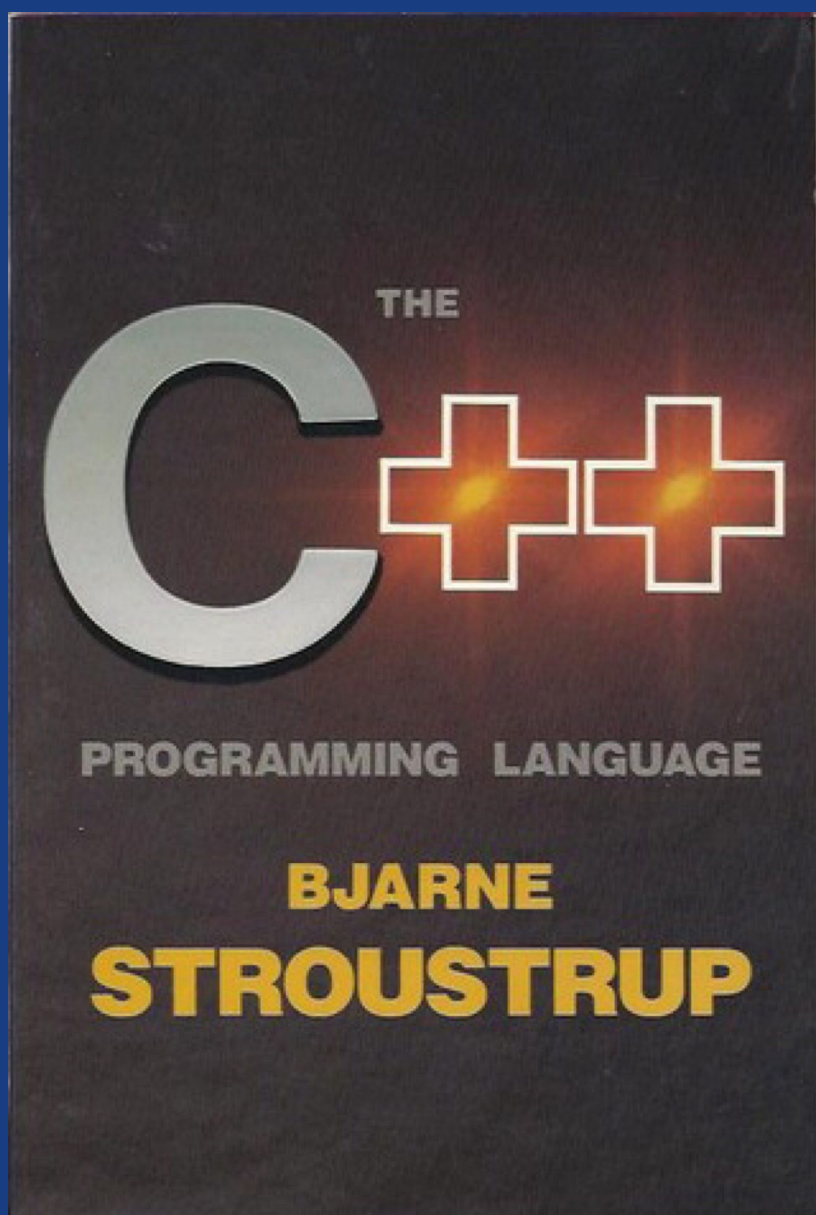
**SIMULA** begin comment airport departure;  
set q counter, q fee, q control, lobby (passenger);

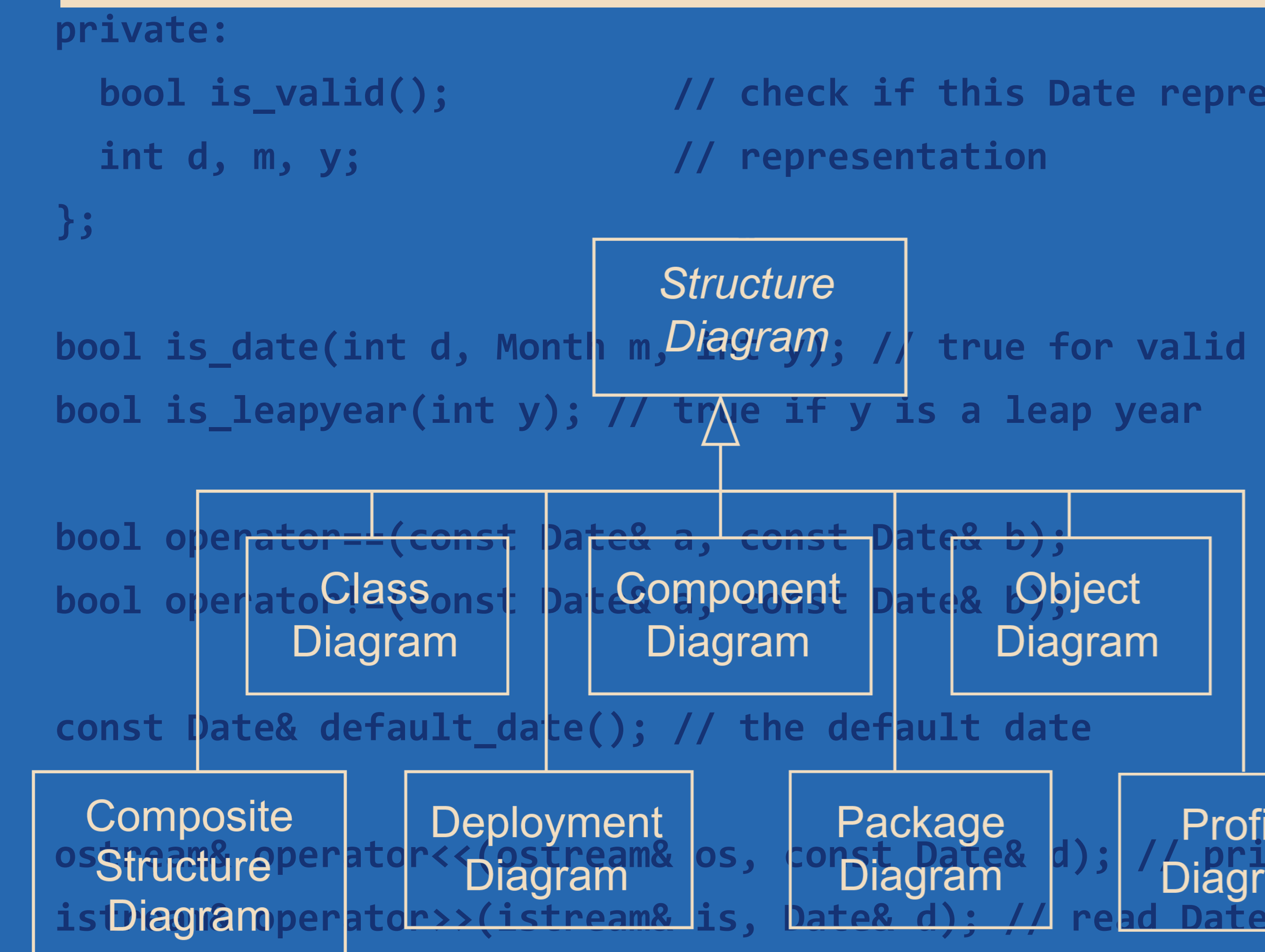
1960s—Simula anticipates many aspects of later object-oriented languages in the context of mathematical simulations.



```
...  
end of SIMULA;  
  
exampleWithNumber: x  
|y|  
true & false not & (nil isNil) ifFalse: [self halt].  
y := self size + super size.  
#($a #a 'a' 1 1.0)  
do: [:each | Transcript  
    show: (each class name);  
    show: (each printString);  
    show: ' '].  
  
namespace Chrono {  
    ^ x < y  
    enum class Month { jan=1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec }  
    class Date {  
    public: // public interface:  
        class Bad_date { }; // exception class  
        Date(int d, Month m, int y) : d(d), m(m), y(y) {}  
        // nonmodifying functions for examining the Date.  
        int month() const; // month  
        int year() const; // year  
        string rep() const; // representation  
        void set(int d, Month m, int y);  
        char_rep(char s[], int n) const; // style string  
        (modifying) function void next();  
        void print();  
    };  
    add_year(int n); // add_year  
    add_month(int n); // add_month  
    private:  
        bool is_valid(); // check if this Date represents a date  
        int d, m, y; // representation  
    };  
    bool is_date(int d, Month m, int y); // true for valid date  
    bool is_leapyear(int y); // true if y is a leap year  
    bool operator==(const Date& a, const Date& b);  
    bool operator!=(const Date& a, const Date& b);  
    const Date& default_date(); // the default date  
    Date& operator<<(ostream& os, const Date& d); // print d to os  
    Date& operator>>(istream& is, Date& d); // read Date from is into d
```

1980s—Object-oriented programming becomes a household name. C++ and later Java focus on practical engineering concerns.

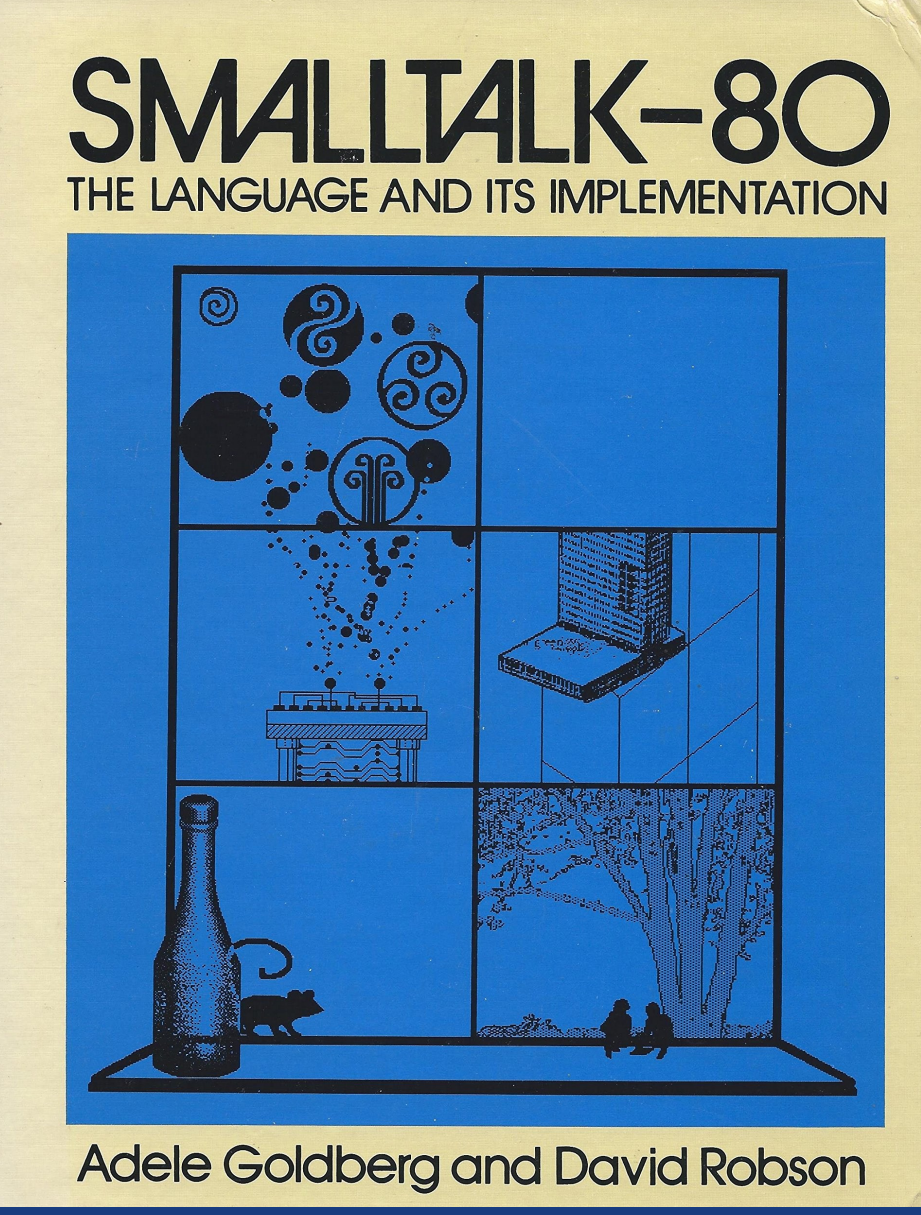



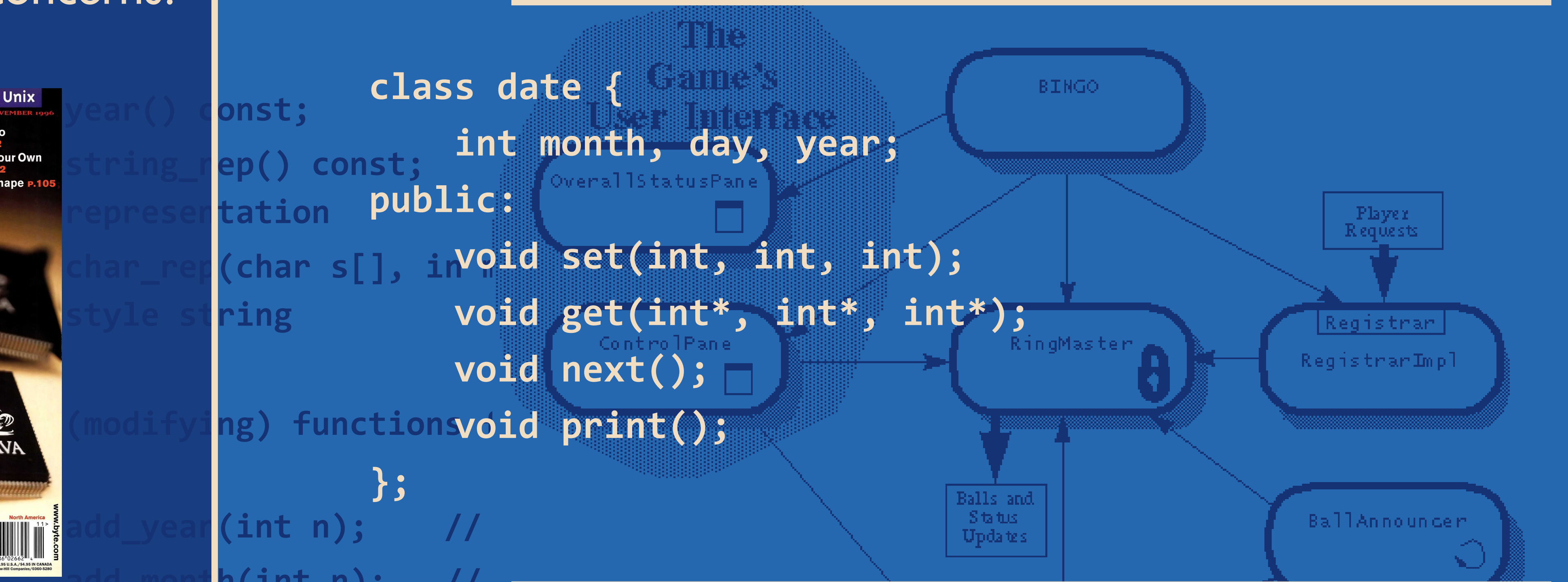


# Object-oriented programming

```
exampleWithNumber: x  
"This is a small method that illustrates every part of Smalltalk method  
syntax except primitives, which aren't very standard. It has unary, binary,  
and key word messages, declares arguments and temporaries (but not blocks  
(literals), accesses a global variable (but not an instance variable, ses  
literals (array, character, symbol, string, integer, float), uses the pseudo  
variable true false, nil, self, and super, and has sequence, assignment,  
return and cascade. It has both key arguments and key word blocks. The  
doesn't do anything useful though"  
if & false not & (nil isNil) ifFalse: [self halt].  
self size + super size.  
#a 'a' 1 1.0)  
[:each | Transcript  
show: (each class name);  
show: (each printString);  
show: ' '].  
< y  
Behaviour Diagram  
Activity Diagram  
State Machine Diagram  
Interaction Diagram  
Use Case Diagram  
Communication Diagram  
Interaction Overview Diagram  
Sequence Diagram  
Timing Diagram  
UML
```

1970s—In the humanistic vision of Smalltalk, objects become the basis of a new medium for communication with the computer.





1990s—UML and new software development processes shift attention to the managerial problem of structuring systems and teams.

