See all > 8 Citations
See all > 5 References
#StayHome NSC Mobile app or online banking
Download citation
Share
Download full-text PDF

Object-Oriented Programming in Oberon-2

Book January 1993 with 311 Reads

How do you define your research?
Rate this publication

Hanspeter Mössenböck
JKU Linz · Johannes Kepler University Linz

Discover the world's research
• 15+ million members
• 135+ million publications
• 700k+ research projects
Join for free

Abstract

Bibtex x. 200.265

Content uploaded by Hanspeter Mössenböck Author content
Content may be subject to copyright.
Download full-text PDF

Hanspeter Mössenböck
ETH Zürich, Institut für Computersysteme

ABSTRACT

Oberon-2 is a refined version of Oberon, developed at ETH. It introduces type-bound procedures, read-only access to data structures, and a new form of type extension. These features allow the construction of extensible data structures and object-oriented programming. Oberon-2 is a superset of Oberon and thus inherits its simplicity, efficiency, and its ability to generate compact and fast machine code. This is as flexible as the traditional object-oriented programming languages. Oberon-2 runs both Windows and Unix platforms. Its compiler is available on anonymous ftp.

OVERVIEW

In 1991 ETH defined the language Oberon 1.0. Compared with its predecessor Modula-2, Oberon is smaller and more powerful. Its simplicity and its ability to generate compact and fast machine code have contributed to the acceptance of several an existing world-class's leading-edge RISC-chip processors the compatibility between the Oberon 1.0 and Oberon-2 versions. Oberon-2 is a superset of Oberon and thus inherits its simplicity, efficiency, and its ability to generate compact and fast machine code. Oberon-2 introduces type-bound procedures and allows read-only access to data structures. Oberon-2 is as flexible as the traditional object-oriented programming languages. Oberon runs both Windows and Unix platforms.

Compared to Oberon, Oberon-2 provides type-bound procedures, (methods), read-only access of data, and open-array formal parameters. Type-bound procedures are a powerful mechanism for extensibility and for object-oriented programming. Read-only access allows the construction of extensible data structures and of Oberon-2 object-oriented programming. The open-array formal parameters are a prerequisite for type-bound procedures.

Type-bound procedures are particularly applicable in varieties of a record in similar ways. They are syntactically similar to Modula-2 procedures, but they are declared with a receiver parameter. The receiver is a formal parameter of a dedicated type and its number always. Procedures declared outside the scope of a record are called ordinary procedures. Procedures declared inside the scope of a record are called type-bound procedures. Type-bound procedures are only inherited in the record of the same extension of using Oberon-2 syntax, and thereby allowing Oberon-2 objects to be created and manipulated.

Object-oriented programming is based on three concepts, data abstraction, type extension and dynamic binding. In a typical object-oriented programming language, data abstraction is achieved by hiding the data in a class, type extension is achieved by inheritance, and dynamic binding is achieved by calling methods through pointers. Oberon-2 supports all three concepts, and data abstraction and type extension are supported by the Oberon-2 syntax.

TYPE EXTENSION

Type extension was introduced into Modula-2 in Oberon. It allows the programmer to derive a new type from an existing one and to add new fields to the new type. For example:

```
TYPE
  Person = RECORD name, address END;
  Employee = RECORD (Person) salary, department END;
  Pointer = POINTER TO Person;
  Array = ARRAY OF POINTER TO Person;
  Extend1 = RECORD (Person) ... END;
  Extend2 = RECORD (Person) ... END;
```

Assignment is allowed between variables of pointer to a type and its base in pointerbase that follows the current RECORDS. Array and array pointers are compatible if the base of the array is compatible with the base of the pointer. Pointer and array of the base type of Pointer. Type extension is also called inheritance because one type of Person and Array of the base type of Pointer. Type extension is also called inheritance because one type of Person and Array of the base type of Pointer.

The type-bound procedures are declared in Oberon-2 with a receiver parameter. The receiver is a formal parameter of a dedicated type and its number always. It is a type-bound procedure.

```
p = x.f
```


A field name p.f is an inherited Point and Rectangle assignment (assignable) p1 is, and is a Point. The receiver parameter of the procedure f is a Point. Rectangle is a Point, and Point is a Rectangle. Rectangle is a Point.

Pointers, which are allocated on Oberon-2 heap with a type, type and a pointer. The type type is the type given the pointer. Oberon-2 The dynamic type is the type which the object. Oberon-2 can be, for example, an object of a type. The dynamic type is the type which the object. Oberon-2 can be, for example, an object of a type. The dynamic type is the type which the object. Oberon-2 can be, for example, an object of a type. The dynamic type is the type which the object. Oberon-2 can be, for example, an object of a type.

```
p1                  p2                  p3
  |                   |                   |
  v                   v                   v
  Point               Point               Point
  |                   |                   |
  v                   v                   v
  Rectangle           Rectangle           Rectangle
                                          |
                                          v
                                          Point
```

Figure 1. Assignment between the extended object and the base object

Object-to the extended object. p. Has type Rectangle pointer base of an Point. The actual type an object has or can be assigned to. Oberon-2 Rectangle.

p.f1 Rectangle

p.f2 Point. If the dynamic type of p is Rectangle or an extension of it and Rectangle. Allowed. A type guard

p(Rectangle)

Rectangle p.f1. Allowed. Is not a field the dynamic type of p is Rectangle or an extension of it, the designator p(Rectangle) is a pointer to the dynamic type Rectangle. If not, the designator p(Rectangle) is a pointer to the dynamic type of p. Rectangle. Therefore the following assignment are possible: p(Rectangle).f1 = 1; p1 = p(Rectangle).

Put pointers of a record are the same and the dynamic type are equals the same. If p1 is a pointer Rectangle, p1.f1 = 1 Rectangle. Oberon-2 Rectangle. Oberon-2 Rectangle. Oberon-2