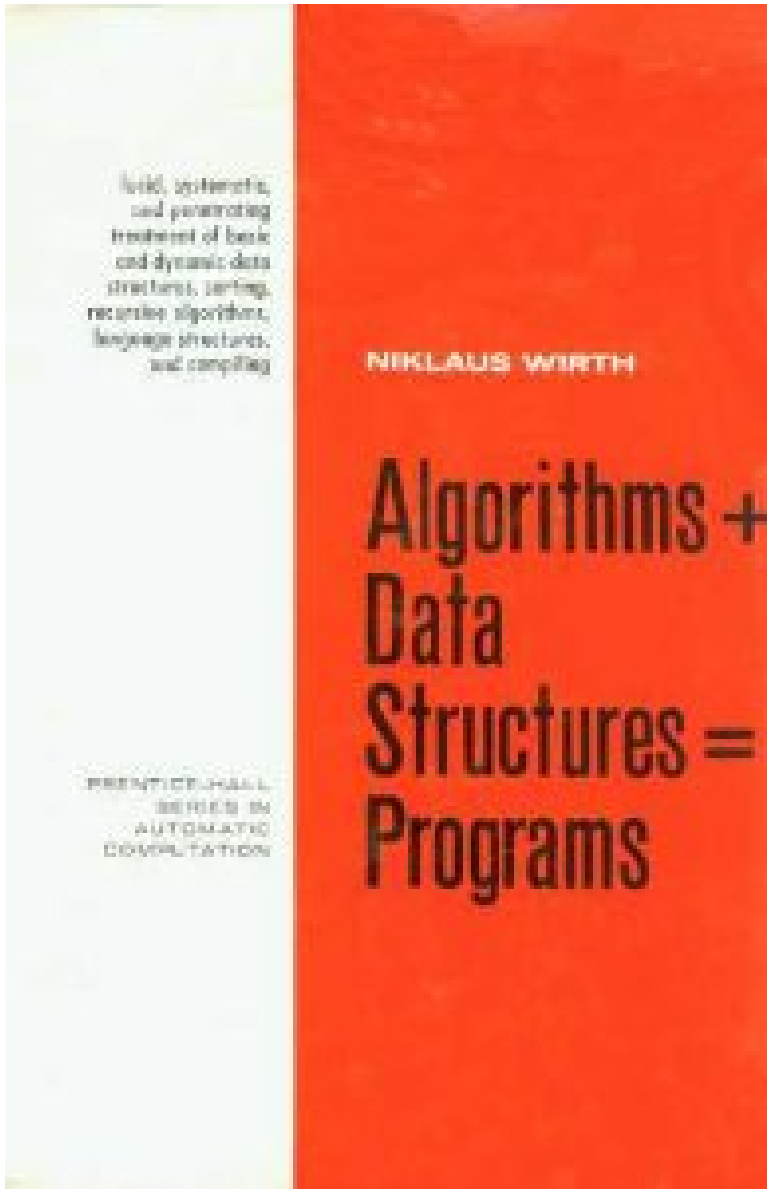
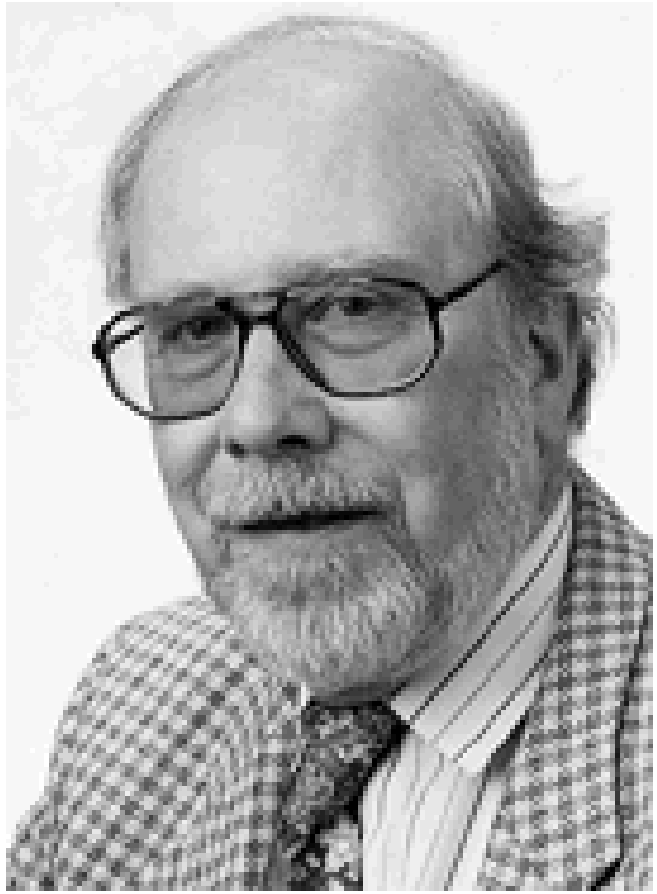


Published in 1976, this book was one of the most influential computer science books of the time, and was used extensively in education.

Niklaus Emil Wirth (1934–)

Ph.D 1963 from Berkeley in Electrical Engineering "A generalization of ALGOL", Professor of Computer Science at ETH Zurich from 1968 until he retired in 1999. In 1984 he received the ACM Turing Award and in 1987 the Computer Pioneer award from the IEEE. Algol committee, ALGOL W, Pascal, Euclid, Modula, Oberon.

1984 Turing Award



Niklaus Emil Wirth (1934–)

For developing a sequence of innovative computer languages,
EULER, ALGOL-W, MODULA and Pascal

Trilingual Pun: Pascal/English/German

When asked how to pronounce his name, he is said to have answered that if you call him by name, it is “virt” (the German pronunciation), and if you call him by value, it is “worth” (the American pronunciation). (“Wert” in German means “worth.”)

Wirth's Law

Wirth's Law asserts that software execution is slowing down faster than hardware is speeding up. "Groves giveth, and Gates taketh away." That is, as the speed of calculation rises, thanks to Intel's Andy Grove, the amount of calculation needed to do the job rises also, thanks to Microsoft's Bill Gates, leaving hardly any gains for the user to enjoy. Some assume that the ignorant or tolerant user is responsible. ETH Zurich's Niklaus Wirth, who actually credits the law to former IBM Research scientist Martin Reiser, claims that software companies' pressure to roll out new products, not user tolerance, is chiefly responsible for feature bloat.
Wirth's article crediting Reiser, "A Plea for Lean Software," appeared in *IEEE Computer*, February 1995, volume 28, number 2.

- Procedural Abstraction
- Data Abstraction

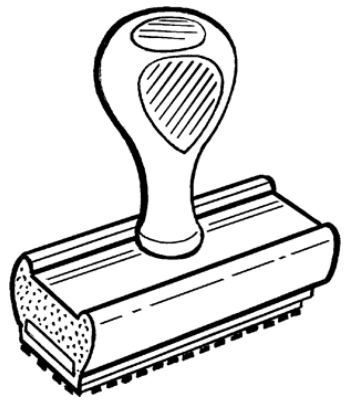
Classes

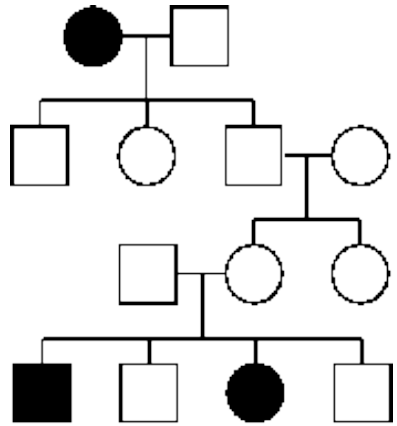
Java is an object-oriented language. It is impossible to give a simple explanation for what that means. Java is organized around a construct known as a class.
What is a class? It is the most prominent feature of the Java language for incorporating all program parts, for creating instances of data structures, and for the design of other classes.

The three I's

There are three goals disparate goals for which a Java class can be used.

1. incorporation (cue `static`)
2. instantiation (cue `new`)
3. inheritance (cue `extends`)





Classes I

The class in Java is fundamental unit of program construction.
And it has syntax:

```
class ClassName {
    // Contents of class:
    //    static fields
    //    static methods
    //    static initializer blocks
    //    static inner classes
}
```

The class is the principle unit of compilation.
These members of the class are said to be “static”, “class”, or “class-wide” members.

Classes II

A Java class is (also) a template for creating new instances called objects.

```
class ClassName {
    // Instance fields
    // Constructors
    // Instance methods
}
```

These members of the class are said to be "instance" members. Instances are constructed using the keyword `new` followed by the name of the class. Arguments are permitted in construction, provided an appropriate constructor has been defined.

Two types: data structures and simulation objects

Classes III

Java classes can be used to derive other classes. This permits data structures to be organized to take advantage of their commonality (if any). Cues: the keyword `extends` and the related keyword `implements`.

```
class ClassName extends SuperClass {
    // Changes and additions to the super class
    //     additional member fields and methods
}
```

Syntax

A class declaration begins with the word `class`:

```
<class declaration> ::=
    "class" <identifier>
    "{" <class body> "}"

<class body> ::= { <class body declaration> }
<class body declaration> ::= ";"
   / "static"  <statement block>
   / { <modifier> } <member declaration>
<member declaration> ::= <field declaration>
   / <method declaration>
   / <constructor declaration>
   / <interface declaration>
   / <class declaration>
```

Note that classes can be nested and that they may contain static initialization blocks.

Syntax

A class declaration may be prefixed by a number of modifiers (some meaningful only for inner classes):

class modifiers	
`public`	
`protected`	
`private`	
`abstract`	incomplete, uninstantiable
`static`	
`final`	all methods final, no subclasses
`strictfp`	all methods, operations FP-strict

Syntax

```
<class declaration> ::=
    "class" <identifier>
    [<type parameter list>]
    ["extends" <type>]
    ["implements" <type list>]
    "{" <class body> "}"
```

Incorporation

Some examples of incorporation from the Java API:

For example, the math functions

```
class Math {
    static double PI
    static int    abs()
    static double sqrt()
    static double atan()
    static double pow()
}
```

For example, the standard I/O package:

```
class System {
    static InputStream in
    static OutputStream out
    static OutputStream err
}
```

Classes include facilities; your cue is the keyword static.

Incorporation

```
class Arrays {
    static void sort ()
    static String toString()
}

class Integer {
    static int parseInt()
    static String toString()
}
```

For example,

```
class MyClass {
    static final int PARAMETER    // ... static memb
    public static void main       // ... point of er
}
```

Incorporation

How are these facilities accessed?

⟨*class name*⟩ . ⟨*static member name*⟩

```
double pi = Math.PI;
double x = Math.sqrt (3.14159);
Arrays.sort (new int[] {4,2,8,1,9,2,7,6,5,8});
int i = Integer.parseInt ("3");
String s = Integer.toString (3, 8);
```

Classes are sometimes used for incorporation (though more often all uses are mixed together).

- `incorporation/Junk.java`
- `incorporation/Main.java`
- `incorporation/Supervisor.java`
- `misc/Init.java` (static initialization blocks)

Intantiation

A class as a template stamps out new data. Each instance has its *own* data as opposed to shared (static) data and as opposed to local data in a method.

Data Structures

Data structure. A **data structure** is a way of organizing and accessing data.
We have seen integers, strings, streams, scanners, arrays, lists, and so on.
It is important to distinguish between the data *structure* (the organization), and the particular data (the instance).
In Java, the class is used as a model or a template for organizing data, and an instance of a template is obtained using the keyword `new`. Instances of classes are also called objects.

- `misc/Main.java` (object creation and method invocation)

How are these facilities accessed?

$$\langle \textit{class instance} \rangle \texttt{.} \langle \textit{instance member name} \rangle$$

Using class instances.

- `misc/Main.java`
- `io/CopyText.java`
- `string/StringFest.java`

- Diversion to lexicographic order?
- Section 3.1 of textbook??

Class as Data Structure

```
public class SimpleTime {
    int hours;
    int minutes;
}
```

One top-level class per file or compilation complications.

- `basic/SimpleTime1.java` and `basic/TimeMain.java`
- `basic/SimpleTime2.java`
- `basic/SimpleTime3.java`

van der Linden, Chapter 2: The Story of O.

- constructors
- constructor chaining
- blank-finals
- immutable classes
- singleton pattern (factory methods) [discussed with collators in lexicographic ordering]
- classes can be nested (use keyword static)

In addition to classes for immutable data structures, classes are used to create objects of simulation. The data structures have state that changes during the lifetime of the object.

Class as Simulation

- `draw/Image.java`
- `class/BankAccount.java`. `assert` statement, preconditions, postconditions
- `simulation/Aircraft.java`

A precondition is a requirement that the caller of a method must meet. If a method is called in violation of a precondition, the method is not responsible for computing the correct result.

Horstman, page 293.

A precondition is an assertion that is guaranteed to be true after a method is called.

A class invariant is an assertion true when a class is constructed and after all methods.

[Go to other next PDF: objects]