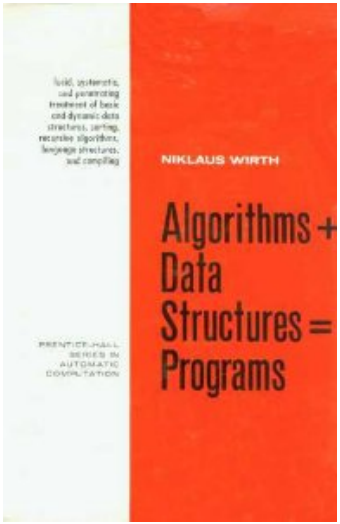
Controlling actions (assignment statements) with `if`, and iteration
if only half the story

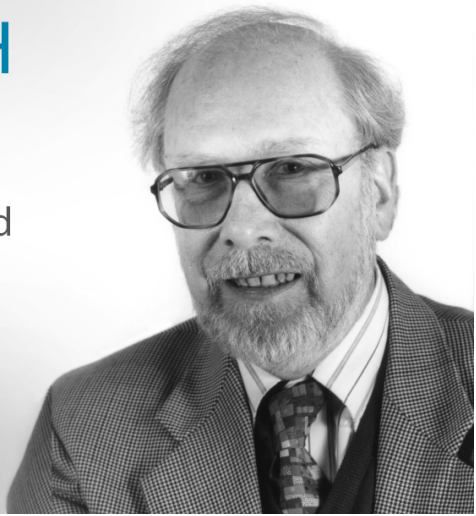


Published in 1976, this book was one of the most influential computer science books of the time, and was used extensively in education.

NIKLAUS WIRTH

Developed EULER, ALGOL-W, MODULA and PASCAL languages

A.M. TURING 1984 AWARD



Niklaus Emil Wirth (1934–2024)

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.

Known for his pioneering research in programming languages and algorithms, Wirth was the recipient of the 1984 ACM A.M. Turing Award, the IEEE Emanuel R. Piore Award, and the Marcel Benoist Prize, among other honors. The chief designer of several programming languages: ALGOL W, Pascal, Euclid, Modula, Oberon.

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). (The German word "Wert" means "value" in English.)

In response to the surprising call-by-name parameter passing in Algol-60, Wirth introduced call-by-value in the Pascal programming language. Call-by-value is the only parameter passing mechanism in Java; C and C++ have call-by-reference in addition.

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. Object-oriented languages try (unsuccessfully) to make everything simple by making everything a *class*. The programmer has but one tool and must keep in mind the different objectives (not just data creation).

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.

> *"Abraham Maslow once said that to him who has only a hammer, the whole world looks like a nail," said Joseph Weizenbaum, a professor of computer science at M.I.T.*

4 April 1982, *New York Times*, Computers Alter Lives of Pupils and Teachers by Edward B. Fiske

In 1966 the prominent psychologist Abraham Maslow published "The Psychology of Science: A Reconnaissance".

Java classes are complicated. However, creating simple data structures has been made easier by the introduction of a kind of Java class called a record in Java. Data abstraction is important, not hard, so we illustrate creating data in Java.

- Main.java
- DB.java

Data design is so important that it is studied in a data structures course. To be prepared to do that, we need to introduce generics and interfaces in Java. But first we examine Java classes in detail.

```
public record Point(double x, double y) {}
```

The output of `javap -p Point`:

```
public final class Point extends java.lang.Record {
  private final double x;
  private final double y;
  public Point(double, double);
  public java.lang.String toString();
  public final int hashCode();
  public final boolean equals(java.lang.Object);
  public double x();
  public double y();
}
```

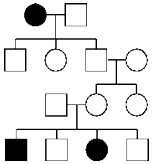
The Three I's

There are three 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 inner classes
    //      [records and enums are implicitly static]
    //    static initializer blocks
}
```

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

class modifiers	
`public`	unrestricted (access modifier)
`protected`	restricted (access modifier)
`private`	local only (access modifier)
`abstract`	incomplete, uninstantiable
`static`	independent
`final`	all methods final, no subclasses
`strictfp`	all methods, operations FP-strict (obsolete)

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()
}
```

Incorporation

Some examples of incorporation from the Java API:
For example, the standard I/O package in the `System` class:

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

Classes include facilities; your cue is the keyword static.

Incorporation

Other examples:

```
class Arrays {
   static String toString()
   static <T> T[] copyOfRange()
   static void sort ()
   static void binarySearch ()
}

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

Note: wrapper classes are used as templates for data, as well.

Incorporation

Also, classes very often hold (incorporate) the entry point.

```
class MyClass {
   static final int PARAMETER  // ... static member field
   static void subProcedure () // ... static subprocedure
   public static void main     // ... point of entry for OS
}
```

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 sometimes all uses are mixed together).

- Junk.java
- Main.java
- Supervisor.java
- Init.java (static initialization blocks)

Class Design

Definition (Cohesion)

A Java class has cohesion if all the members constitute a unified, easily identified purpose.

II: Instantiation

Instantiation

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.

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

Access of Instance Members

How are objects (instances of classes) created?

Definition

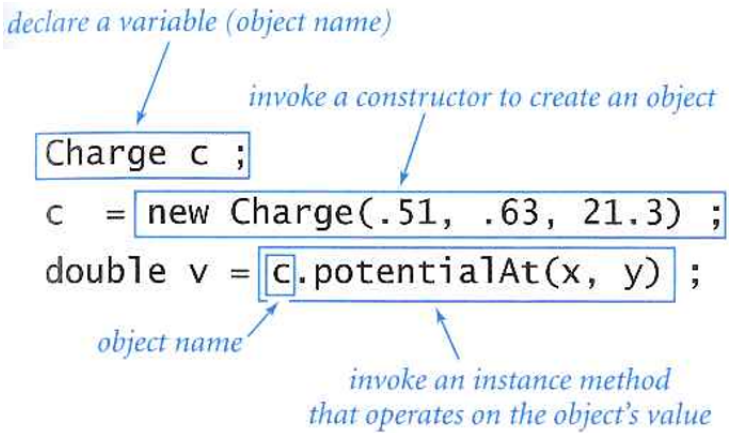
A constructor in Java is a class member with special syntax that is used to create an instance of a class (in the heap) and it usually initializes the attributes.

Access of Instance Members

How are these facilities accessed?

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

```
String line = stdin.nextLine();
int length = "abc".length();
```


declare a variable (object name)
invoke a constructor to create an object
Charge c ;
c = new Charge(.51, .63, 21.3) ;
double v = c.potentialAt(x, y) ;
object name
invoke an instance method
that operates on the object's value

Using class instances.

- `Main.java`
- `CopyText.java`
- `StringFest.java`

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
- basic/SimpleTime4.java
- basic/SimpleTime5.java
- basic/SimpleTime6.java
- basic/SimpleTime7.java

Hiding Declarations

- `basic/Hide.java` – illegal
- `basic/Eclipse.java` – legal, but style error

Java 16 Records

- basic/SimpleTime8.java
- basic/SimpleTime9.java
- basic/SimpleTimeA.java
- basic/SimpleTimeB.java
- basic/SimpleTimeC.java

Simple, complete classes as data structures.

- `class/Point2DR.java` `class/Point2D.java` [`this`; constructor chaining]
- `class/CircleR.java` `class/Circle.java` [composing data]
- `class/LineR.java` `class/Line.java`
- `class/Polynomial.java` [Arrays need private access.]
- `class/Person.java` [Mutable objects need private access], `class/Person8.java`
- `class/Elephant.java` [Private constructors are useful.]

Summary

- classes can be nested (use keyword static)
- constructors
- private constructors
- constructor chaining
- blank-finals
- immutable classes
- singleton pattern (factory methods)
- eclipsing declarations

Recursion

- basic/Body1.java
- basic/Body2.java
- basic/Body3.java
- basic/Body5.java
- basic/Body6.java
- basic/Body7.java

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.

- basic_class/Cell1.java
- basic_class/Cell2.java
- basic_class/Cell3.java
- basic_class/Cell5.java
- basic_class/Cell6.java

Class as Simulation

- `draw/Image.java`
- `class/BankAccount.java`. `assert` statement, preconditions, postconditions
- `assert/GeoPoint.java`. class invariants
- `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]