Generic programming: C++ vs. Java

Erich Kaltofen
North Carolina State University
www.math.ncsu.edu/~kaltofen

Joint work with: Laurent Bernadin (Waterloo Maple)
Bruce Char (Drexel University)
LinBox project members

2nd Edition
Ignores Java 1.1

JAVA IN A NUTHOUSE

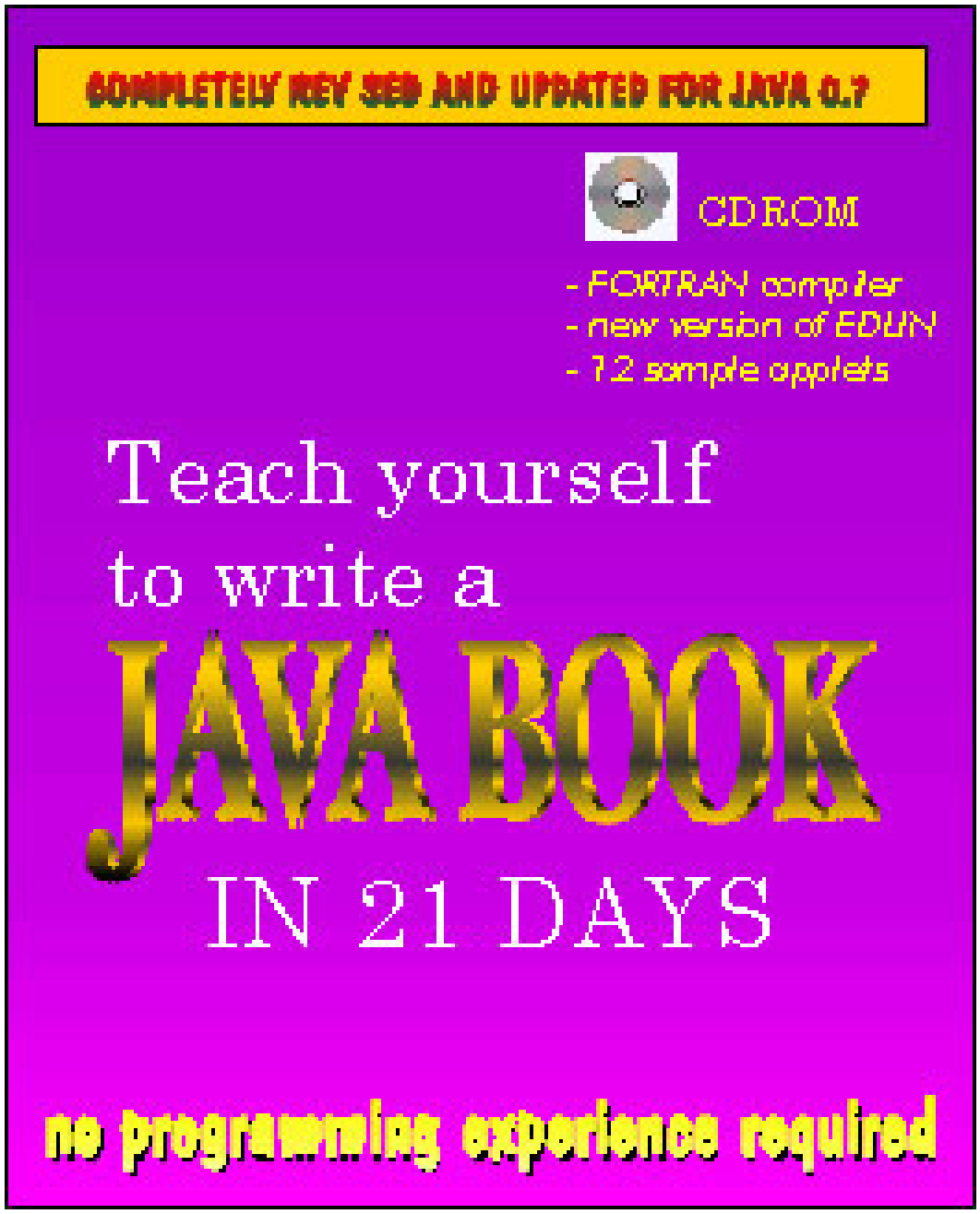
A Desktop Quick Irreverence to Microsoft Java

O'REILLY

Kevin McCurley

Java advantages

- GUI
- Distributed computing (Schreiner 99, Masdis)
- Visual programming (MathBeans)
- Black-box functions (Sandbox, Melissa)
- Serialization standard (PDG Openmath)
- Standard libraries
- Platform independence



(Kevin McCurley)

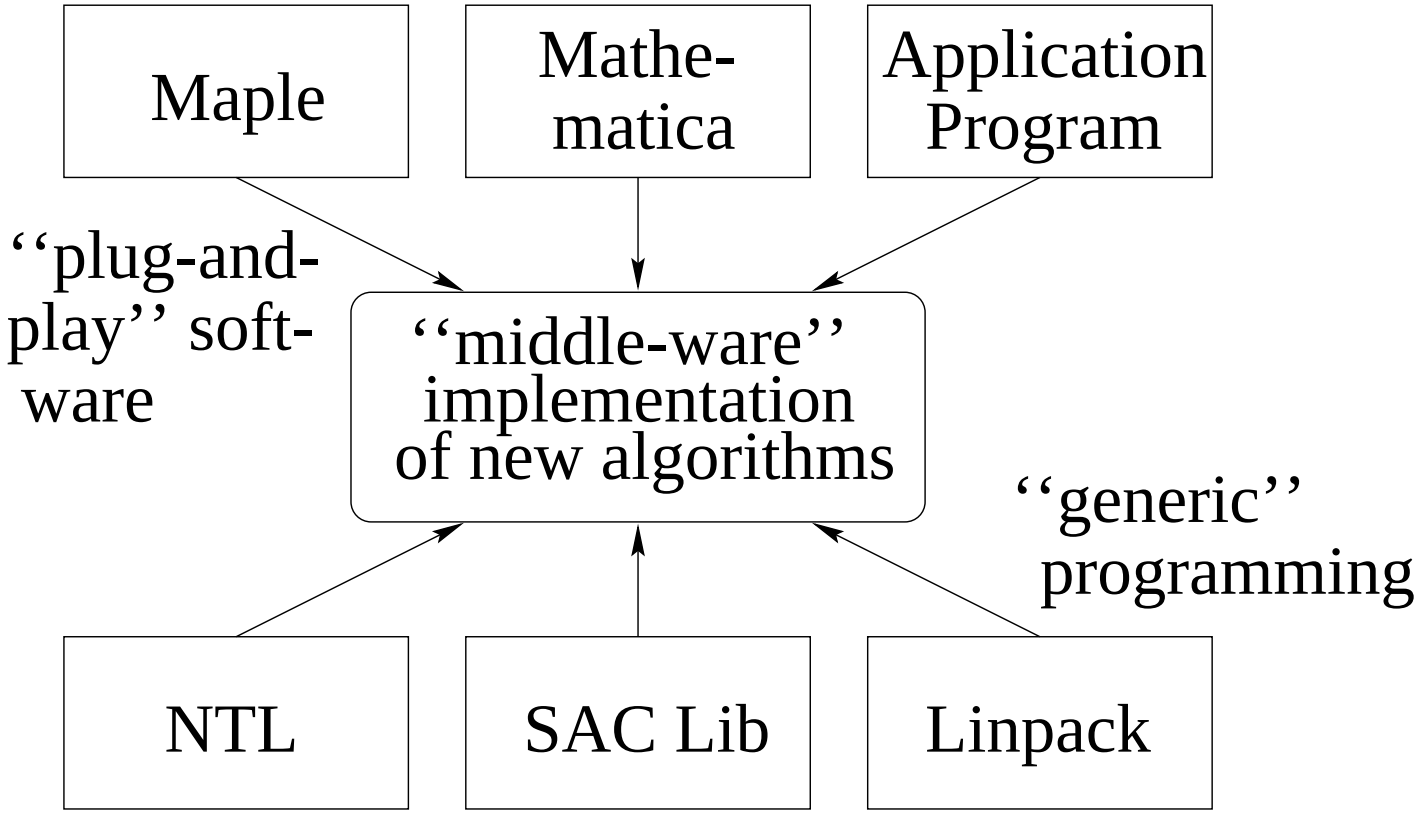
Is Java fit for large-scale computing?

- Large and complex programs

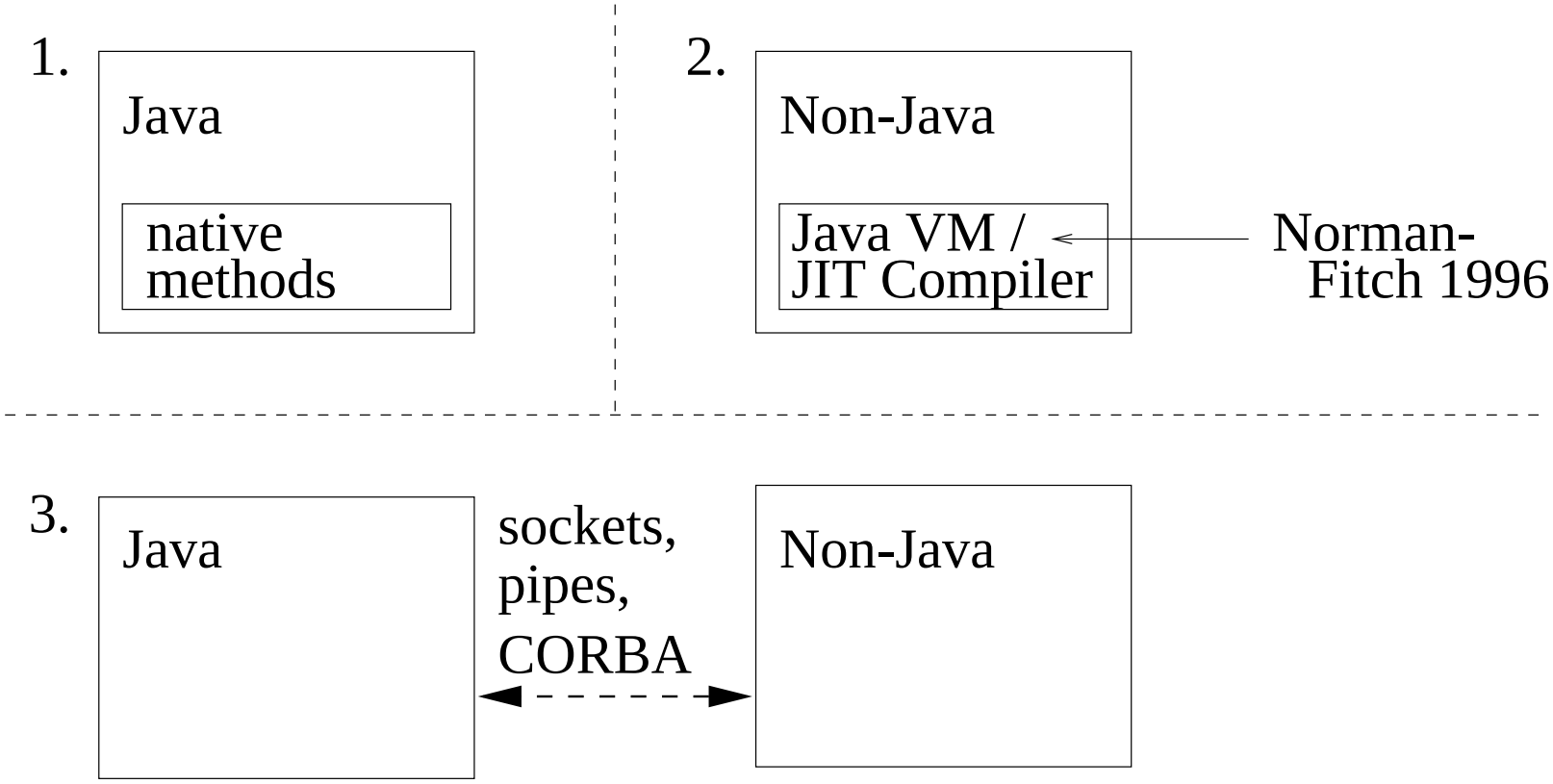
- High performance calculations

Component-wise system design

Plug-and-play components ($\longrightarrow$conglomerates, PSEs)



Interfaces between Java/non-Java components



Generic Programming

- static (compile-time) binding: templates (C++, GJ, NextGen)
  performance
- dynamic (run-time) binding: interfaces (Java)
  Does C++ need interfaces?
  Abstract base classes, virtual member funs, and multiple inheritance (almost) do the trick.
- C++-style templates: GJ/NextGen
  Are templates really needed?
- STL-style allocators as interface “glue” for divergent storage models
- Algorithmic shortcuts into the basic modules
  partial template specialization

toString with non-standard C++

```
#include <iostream>
#include <string>
#include <strstream>

template <class T>
string& toString(const T& x) {
  ostrstream ost;
  ost << x;
  return *(new string(ost.str()));
} // toString()
```

toString in Java

```
class A {
...
public void print(PrintWriter os)
     { // can supply either a   new PrintWriter(System.out)
       //             or a      new PrintWriter(new CharArrayWriter())
       //                       (see toString below)
       os.print("write A obj to stream"); ...
     }// end print

public String toString()
     { // with above print method,
       // toString always can be implemented like this
       CharArrayWriter buf = new CharArrayWriter();
       PrintWriter os = new PrintWriter(buf);
       print(os);
       return buf.toString();
     }// end toString

}// end class A
```

toString with standard C++

```
#include <iostream>
#include <string>
#include <sstream>

template <class T, class Ch = char>
basic_string<Ch>& toString(const T& x) {
  basic_ostringstream<Ch> ost;
  ost << x;
  return *(new basic_string<Ch>(ost.str()));
} // toString()
```

Abstract Field

```
class abstract_field {
public:
  field_base* elem_ptr;

  abstract_field ( field_base* init_elem_ptr = 0);

  class element {
  public:
    field_base* elem_ptr;
    element(const abstract_field&, int = 0);
    element(const element&);
    ~element();
    element& operator=(const element&);
  }; // class element

  bool equal(const element& x, const element& y) const;
  element& add(element& x, const element& y, const element& z) const;
  ...

  void print(ostream& os, element& x) const;
  void read(istream& is, element& x) const;
}; // class abstract_field
```

```
class field_base {
  friend istream& operator>>(istream&, abstract_field::element&);
  friend class abstract_field;

public:
  virtual bool operator==(const field_base&) const = 0;
  virtual field_base& operator+(const field_base&) const = 0;
  ...

  field_base () {}
  field_base (const field_base& fb) = 0;
  virtual field_base* clone() const = 0;
  virtual field_base* make_int(int) = 0;
  virtual field_base& operator=(const field_base&) = 0;
  virtual ~field_base() {}

  virtual void print(ostream& os) const = 0;
  virtual void read(istream&) = 0;
}; // end class field_base
```

Implementation for abstract field elements

```
inline abstract_field::element::element(const abstract_field& af,
                                        int x = 0 )
{ elem_ptr = af.elem_ptr->make_int(x); }

inline abstract_field::element::element(const element& x)
{ elem_ptr = x.elem_ptr->clone(); }

inline abstract_field::element::~element()
{ delete elem_ptr; }

inline abstract_field::element& abstract_field::element::operator=
(const abstract_field::element& x) {
  if (this != &x) {
    delete elem_ptr;
    elem_ptr = x.elem_ptr->clone();
  }
  return *this;
}
```

Implementation for abstract fields

```
inline abstract_field::abstract_field ( field_base* init_elem_ptr = 0) :
  elem_ptr (init_elem_ptr) {}

inline bool abstract_field::equal(const element& x,
                                  const element& y) const
{ return (*x.elem_ptr == *y.elem_ptr); }

inline abstract_field::element& abstract_field::add(
                    abstract_field::element& x,
                    const abstract_field::element& y,
                    const abstract_field::element& z) const {
  x.elem_ptr = &( (*y.elem_ptr) + (*z.elem_ptr) ) ;
  return x;
}
```

```
public interface Ring {
        public interface Element {
                public boolean iszero();
                ...
                }
        public Element fromInteger(int n);
        public Element add(Element a, Element b);
        public Element multiply(Element a,Element b);
        ...  }


public class DensePolynomial implements Ring {
        public class Element implements Ring.Element {
                private Ring.Element[] _coeffs;

                public int degree() {
                        return _coeffs.length-1; }
                ...
                }// end class Ring.Element

        private Ring _R;

        public Element add(Ring.Element a, Element b) { ... }
        }
```

```
Ring R = new Zmod(17);
Ring P = new DensePolynomial(R);
DensePolynomial.Element p = P.add( P.monomial(10),
                                   P.one() );
DensePolynomial.Element q = P.add( P.monomial(3),
                                   P.one() );
p = P.multiply(p,q);
```

Performance considerations

- Garbage collection
- Interpreted bytecode / JIT
- Benchmarks

In-Place Polynomial Arithmetic

- Multiply degree n polynomials over $\mathbb{F}_{17}$
- Dense array of hardware integers
- Java: Sun JDK 1.2 (beta 5)
- Maple: modp1 datastructure, R5
- C: Sun Workshop 4.2, flags: -O
- C*: flags: -native -fast -xO4

n	Java	Maple	C	C*	Java vs. C*
10000	7s	9s	9s	3s	2.33
20000	30s	37s	36s	13s	2.31
30000	69s	77s	82s	31s	2.23
40000	124s	137s	146s	56s	2.21
50000	196s	218s	231s	89s	2.20

Generic Polynomial Arithmetic

- Multiply degree *n* polynomials over a generic ring
- Particular ring is $\mathbb{F}_{17}$ again
- Aldor: 1.1.10b + $\sum^{\text{it}}$
- C++: gcc 2.8.1, templates

n	Java	Aldor	C++	Java vs. C++
1000	2.6s	1.2s	0.9s	2.8
2000	9.1s	4.3s	3.6s	2.6
3000	20s	10s	8s	2.5
4000	36s	19s	14s	2.6
5000	57s	30s	22s	2.6
6000	82s	42s	31s	2.6
7000	111s	56s	42s	2.6
8000	146s	72s	57s	2.6

Conclusions [ISSAC'99 talk]

- Java as component glue and for GUI code
- Performance is improving (compiler technology)
- Template facilities are being missed

- Java library standardization would help