Kava:
A Java Dialect with a Uniform Object Model for Lightweight Classes

David F. Bacon
IBM T.J. Watson Research Center

Java Grande/ISCOPE 2001 Conference
Stanford University
June 2, 2001
The Problem

- Distinction between objects and primitives
  - In C++ and Java, totally non-uniform
  - In Smalltalk, integers look like objects
    » But can’t be defined within the language

- Not orthogonal

- Creates pressure for language changes
  - New primitive types; new operators

- Semantics of primitives hardware-defined
Manifestation: Java Numerics

- Java Grande NWG Requirements
  - Complex arithmetic
  - Lightweight classes
  - Operator overloading
  - Multi-dimensional arrays
  - Exploitation of floating-point hardware
- Solution should be clean and general
  - Not just specific to NWG
- See *Growing a Language* by Guy Steele
Solution: Kava

- Highly
  - efficient
  - extensible
- Extends OO abstraction to the (abstract) bit level
- Eliminates primitive/object distinction
  - Exposes value semantics to programmer
- Allows user-defined primitives
- Backward-compatible Java extension
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- Implementation Strategy
- Advantages of the Kava Approach
- Current Status and Future Work
Enumerations

```java
enum day {
    Monday, Tuesday, Wednesday, Thursday, Friday,
    Saturday, Sunday;
}

day d = Monday;
if (d == Saturday || d == Sunday)
    System.out.println("Let's party");
```
Abstraction: Enumeration Methods

```java
enum day {
    Monday, Tuesday, Wednesday, Thursday, Friday,
    Saturday, Sunday;

    public boolean weekend() {
        return this == Saturday || this == Sunday;
    }
}

day d = Monday;
if (d.weekend())
    System.out.println("Let's party");
```
Enumeration-Based Arrays

```java
int happiness[day] = { 0,1,4,3,7,10,8 };

if (happiness[Monday] < happiness[Saturday])
    System.out.println("Try going sailing!");
```

- Subscripted by enumerations instead of ints
- Fixed size
- Never require run-time bounds checks
- Base case for numeric types
Iteration over Enumerations

- Iteration over values of an `enum` type
- Java `for(;;)` idiom relies on value outside of range
- Use `++` operator on `enum` with `do/while`

```java
    day d = day.first;
    do {
        total += happiness[day];
    } while (day++ != day.last);

    int lifestyle = total/day.size;
```
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- Implementation Strategy
- Advantages of the Kava Approach
- Current Status and Future Work
bit Type Definition

enum bit {
    zero, one;

public bit sum(bit a, bit b) { return sum[this][a][b]; }

static final bit not[bit] = {one,zero};
static final bit and[bit][bit] = {{zero,zero},{zero,one}};
static final bit sum[bit][bit][bit] = {{zero,one},{one,zero}},
    {{one,zero},{zero,one}};

...
**bit** Type: Overloaded Operators

```java
enum bit {
    zero, one;
}

public bit ~ this { return not[this]; }
public bit this & bit that { return and[this][that]; }
public bit sum(bit a, bit b) { return sum[this][a][b]; }

static final bit not[bit] = {one, zero};
static final bit and[bit][bit] = {{zero,zero},{zero,one}};
static final bit sum[bit][bit][bit] = {{zero,one},{one,zero}},
    {{one,zero},{zero,one}};
```

...
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- Implementation Strategy
- Advantages of the Kava Approach
- Current Status and Future Work
Values

- Generalized read-only objects
- Do not allow:
  - synchronization
  - address equality
  - modification outside of constructor
- Efficient
  - small values can be shared by copy (registers)
  - large values can be shared by reference
- Enumerations are a special kind of value
Class Hierarchy

Instance
- toString
- clone
- hashCode
- equals
- getClass

Value

Object
- wait
- notify
- finalize
- notifyAll

Enum
- index
- first
- size
- last
User-Defined ubyte: Data Structure

```c
enum byteindex { b0, b1, b2, b3, b4, b5, b6, b7 };

final value ubyte {
    private bit data[byteindex];

    ...
}
```

---

David F. Bacon  
6/1/2001  
IBM Watson Research Center
ubyte: Less-Than Operator

- Iterates over bit values
- Uses enumeration != and < operators

```java
public boolean this < ubyte that {
    byteindex x = byteindex.last;
    do {
        if (data[x] != that.data[x])
            return data[x] < that.data[x];
    } while (x-- != byteindex.first);
    return false;
}
```
abyte: Complement Operator

```java
public ubyte ~ this {
    return new ubyte..complement(this);
}

private ubyte..complement(ubyte b) {
    byteindex x;
    do {
        data[x] = ~ b.data[x];
    } while (x++ != byteindex.last);
}
```

- Creates new value
- Must use c’tor
  - Named c’tor
- Uses bit ~ op
ubyte: Plus Operator

```java
public ubyte this + ubyte that {
    return new ubyte.sum(this, that);
}

private ubyte.sum(ubyte a, ubyte b) {
    bit carry;
    byteindex x;
    do {
        data[x] = carry.sum (a.data[x], b.data[x]);
        carry    = carry.carry(a.data[x], b.data[x]);
    } while (x++ != byteindex.last);
}
```
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- Implementation Strategy
- Advantages of the Kava Approach
- Current Status and Future Work
Bound and Optional References

- Optional references may be `null`
  - `@Foo x = null; x = new Foo();`

- Bound references are never `null`
  - `Foo y = new Foo();`

- Bound references are initialized by default
  - `Foo z; Foo z = new Foo();`
  - `int x; int x = new int(); int x = 0;`
Class point

final value point {
   public ubyte x;
   public ubyte y;

   public point(ubyte x, ubyte y) { this.x = x; this.y = y; }

   public point this + point that {
      return new point(x+that.x, y+that.y);
   }
   ...  
}

Uniform Object Model

point
- x
- y

ubyte
- data

bit[]
- x0
- x1
- x2
- x3
- x4
- x5
- x6
- x7

bit
- 1
- 0

bit
- 1
- 0
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- **Implementation**
- Advantages of the Kava Approach
- Current Status and Future Work
Implementing the Object Model

Instance ➔
- sync
- hash
- GC info
- class pointer
- field 1 ➔
- field 2 ➔
Implementing the Object Model
Implementing the Object Model
Implementing the Object Model

primitive

field 1 field 2
Space-Efficient Implementation

point

\[
x: \text{ubyte} \\
\begin{array}{cccccc}
  x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
  0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
y: \text{ubyte} \\
\begin{array}{cccccc}
  x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
  0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\text{point}(15,15) \\
0000111100001111
\]

(16-bit halfword)
Implementation Strategy

Enumerations
Values
Value Arrays
Overloading
Bound refs
Casts

Kava Translator
Java bytecode
Standard JVM

Jalapeño KVM
Semantic expand
Enum conversion
Unboxing
Field reorg.

David F. Bacon 6/1/2001 IBM Watson Research Center
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- Implementation Strategy
- Advantages of the Kava Approach
- Current Status and Future Work
Advantages of Kava

- As with other proposals:
  - Provides lightweight types and operator overloading
- Eliminates primitive/object distinction
- Abstraction extends all the way down
  - Semantics of primitives definable inside language
- Can easily add new primitives
  - Can use existing hardware types and operations
    » Or define new ones
  - Value inlining makes them space-efficient
Advantages of Kava

- Accommodates new hardware
  - Prototyping of new hardware operators
  - Inclusion of new hardware operators in language
- Simplifies VM instruction set. Only need:
  » Types: value, array, object
  » Method invocation
- Result: easier to port, spread language (See Steele again)
Examples of Kava Value Types

- complex
- rational
- ipaddress
- date
- day, month

Enumerations
- boolean
- bit

- interval
- decimal
- point
- time
- timestamp
- dna, rna
- addressClass
Satisfaction of NWG Requirements

- Lightweight classes
- Operator overloading
- Complex arithmetic
  - Add complex value class
- Efficient use of floating point hardware
  - Use semantic expansion in virtual machine
- Multi-dimensional arrays
  - Can be added using overloading facilities
Systemic Effects

- Should reduce dynamic allocation
  - Many common objects will become scalar values
  - Strings will only require one object instead of two

- Concurrency issue:
  - “Large” values must be atomically updated
    » Limits size of unboxed values to 32/64 bits
  - Synergy with Guava (Bacon et al. OOPSLA’00)
    » Not a problem if language is race-free
Talk Outline

- Motivation
- Enumerations
- Operator Overloading
- Values
- Object Model
- Implementation Strategy
- Advantages of the Kava Approach
- Current Status and Future Work
Status

◆ Kava-to-bytecode Translator (Jikes)
  – Translates
    » Enumerations
    » Values
  – To do:
    » Bound refs, value arrays, casts, c’tor fns, checking
  – Planned completion September ‘01

◆ Jalapeño VM Optimizations
  – Implementation beginning July ‘01
Future Plans

◆ Kava-to-bytecode Compiler
  – Public release via alphaWorks or public domain
◆ Kava VM
  – Will be available via Jalapeño university license
◆ More information:
  – dfb@watson.ibm.com