llava
Java in Lisp Syntax

Harold Carr

(Sun Microsystems)

carr@llava.org

http://llava.org

llava - Main Point

- **Prefix List version of Java with macros & repl**
- **Not:**
  - **Common Lisp/Scheme/* written in Java**
  - **No need for special notation to access Java**
  - **No disjoint set of types**

```
(package org.llava.demo)

(import java.lang.ArithmeticException)
(import java.lang.Exception)

(let ((bomb 1))
  (define (demo)
    (try
       (if (< bomb 0)
           (throw (new 'Exception "Give up!")))
       (list "Normal result is: " (/ 2 bomb))
     (catch (ArithmeticException e)
       (list "Arith: " e))
     (catch (Exception e)
       (list "Except: " e))
     (finally
       (set! bomb (- bomb 1))))))


(demo) => ("Normal result is: " 2)
(demo) => ("Arith: " ArithmeticException: / by zero)
(demo) => ("Except: " java.lang.Exception: Give up!)
```

Initial Motivation

interactive Java – kawa, skij, silk/JScheme

Java/Scheme interoperability issues motivated me to roll-my-own

provide *direct* access to Java
plus features from Lisp (procedures, macros, lists, symbols)

Current Motivation

make Lisp (llava) transition easy for Java people
(like Java transition easy for C++ people)

Maximum leverage of Java
(IDEs, debuggers, unicode, libraries, JavaDoc, ...)

Outline of Talk

- **Overview of the llava language**
- **llava language design**
- **Brief tour of current implementation**
- **Future work**

```
package org.openhc.llavademo.pb;

import java.util.LinkedList;
import java.util.List;

public abstract class PointBase {
  protected List history =
    new LinkedList()
  protected int x = 0;
  protected int y = 0;
  public int getX() { return x; }
  public int getY() { return y; }
  protected void move(int dx, int dy) {
    x += dx; y += dy; moved();
  }
  protected void moved() {
    history.add(this.toString());
    System.out.println("Moved: " + this);
  }
  public abstract List getHistory() {
    return history;
  }
  public String toString() {
    return getName() +
      " x: " + x +
      " y: " + y;
  }
  protected abstract String getName();
}
```

```
(package org.openhc.llavademo.pb)

(import java.util.LinkedList)
(import java.util.List)

(abstract class PointBase
  (protected List history
    (new 'LinkedList))
  (protected int x 0)
  (protected int y 0)
  (public int (getX) x)
  (public int (getY) y)
  (protected void (move dx dy)
    (+= x dx) (+= y dy) (moved this))

  (protected void (moved)
    (add history (toString this))
    (println (System.out) (+ "Moved: " th

  (public List (getHistory)
   history)

  (public String (toString)
    (+ (getName)
       " x: " x
       " y: " y))

  (protected abstract String (getName)))
```

```
(package org.llava.demo)

(import java.lang.ArithmeticException)
(import java.lang.Exception)

(let ((bomb 1))
  (define (demo)
    (try
       (if (< bomb 0)
           (throw (new 'Exception "Give up!")))
       (list "Normal result is: " (/ 2 bomb))
     (catch (ArithmeticException e)
       (list "Arith: " e))
     (catch (Exception e)
       (list "Except: " e))
     (finally
       (set! bomb (- bomb 1))))))


(demo) => ("Normal result is: " 2)
(demo) => ("Arith: " ArithmeticException: / by zero)
(demo) => ("Except: " java.lang.Exception: Give up!)
```

new and virtual/static method calls

Java

```
import java.util.Hashtable;
Hashtable ht = new Hashtable();
ht.put(“three”, 3);
Byte b = Byte.decode(“3”);
```

llava

```
(import java.util.Hashtable)
(set! ht (new 'Hashtable))
(put ht “three” 3)
(set! b (Byte.decode “3”))
```

new and virtual/static method calls

JScheme

```
(import “java.util.Hashtable”)
(define ht (Hashtable.))
(.put ht “three” 3)
(define b (Byte.decode “3”))
```

abcl

```
(setq ht (jnew (jconstructor “java.util.Hashtable”)))
(jcall (jmethod “java.util.Hashtable” “put”
                “java.lang.Object” “java.lang.Object”)
       ht “three” 3)
(setq b (jstatic (jmethod “java.lang.Byte” “decode”
                          “java.lang.String”)
        nil “3”))
```

virtual/static fields

Java

```
import org.omg.CORBA.IntHolder;
IntHolder ih = new IntHolder();
ih.value =3;
in.value;
import java.io.File;
File.pathSeparator;
```

llava

```
(import org.omg.CORBA.IntHolder)
(set! ih (new 'IntHolder))
(value! ih 3)
(value ih)
(import java.io.File)
(File.pathSeparator)
```

JScheme

```
(import “org.omg.CORBA.IntHolder”)
(set! ih (IntHolder.))
(.value$ ih 3)
(.value$ ih)
(import “java.io.File”)
File.pathSeparator$
```

Reflective Invocation

```
(toLowerCase (substring "Foo-Bar" 4))

   => "bar"
```

Reflective Invocation

skij

```
(define (lastMod name)
  (let ((f (new 'java.io.File name)))
    (if (invoke f 'exists)
        (new 'java.util.Date
             (invoke f 'lastModified)))))

(define (sep)
  (peek-static 'java.io.File 'separator))

(define (lispList name)
  (list (new 'java.io.File name)))

(define (javaList name)
  (invoke (new 'java.io.File name)
          'list))
```

llava

```
(import java.io.File)
(import java.util.Date)

(define (lastMod name)
  (let ((f (new 'File name)))
    (if (exists f)
        (new 'Date
             (lastModified f)))))

(define (sep)
  (File.separator))

(define (lispList name)
  (-list (new 'File name)))

(define (javaList name)
  (list (new 'File name)))
```

llava procedures viz Java method calls

```
(define toLowerCase
  (lambda (x) (toUpperCase x)))


(toLowerCase (substring "Foo-Bar" 4))

   => "BAR"




(define (toLowerCase x)
  (- x))


(toLowerCase (substring "Foo-Bar" 4))

   => "bar"

(toLowerCase 3.4) => -3.4
```

package/import

```
(package org.example)
(import java.text.DateFormat)

(define (newDateFormat)
  (DateFormat.getInstance))

(define (whatZone x)
  (let* ((tz (getTimeZone x))
         (n  (getDisplayName tz)))
    n))


(package some.other.package)
(import org.example)

(whatZone (newDateFormat)) => "Mountain Standard Time"
(whatZone (DateFormat.getInstance)) ; Error: ...
```

macros

```
(define-syntax while
  (lambda (tester . body)
    `(do ()
         ((not ,tester))
       ,@body)))
```

Outline of Talk

- **Overview of the llava language**
- **llava language design**
- **Brief tour of current implementation**
- **Future work**

Language Design - comments

```
// single line
comment
;  ...

/* block
   comment */
(/* ...)
(-comment- ...)

/**
 * JavaDoc comment
 * @author me
 */
(/** @author)
(-doc- ...)


(/* more . . .)

(/* "more . . .")
```

Language Design - identifiers

```
(define *global* 3)

(define (some-procedure 1st-arg 2nd-arg)
  (let ((+sum+ (+ 1st-arg 2nd-arg)))
    (list +sum+ *global* '-foo)))
```

Identifiers for interfaces, classes, fields and methods must follow Java rules.

Language Design - literals

Java keywords: **`public abstract static`** ...

llava keywords: **`define lambda quote let`** ...

Boolean literals: **`true false`** (not **`#t #f`** or **`t nil`**)

Settle design decisions in favor of Java

But, Java characters: **`'a'`** conflict with quote reader macro.

Therefore, llava characters use Lisp representation: **`#\a`**

Language Design - operators

`foo = bar;`	`(= foo bar)`	`(set! foo bar)`
`x == y`	`(== x y)`	`(eq? x y)`
`x && y`	`(&& x y)`	`(and x y)`
`x != y`	`(!= x y)`	`(not (eq? x y))`
`!x`	`(! x)`	`(not x)`
`x += 1`	`(+= x 1)`	`(set! x (+ x 1))`

types, values, variables

```
(import java.util.Hashtable)
(import java.util.Map)

(public class Table
  (private static int numCreated 0)
  (private Map table)

  (public (Table)
    (+= numCreated 1)
    (= table (new 'Hashtable)))

  (public static int (numCreated) numCreated)

  (public void (put (String key) (int value))
    (put table key value))

  (public int (get (String key))
      throws NoSuchElementException
             MinusOneException
    (let ((v (get table key)))
      (cond ((== v null)
             (throw (new 'NoSuchElementException)))
            ((< v 0)
             (throw (new 'MinusOneException)))
            (else
             (intValue v))))))
```

Language Design - fields

```
protected int numCreated = 0;

(protected int numCreated = 0)

(protected numCreated = 0)
(protected = numCreated 0)
(protected int = numCreated 0)
(protected = int numCreated 0)
(= protected int numCreated 0)

(protected numCreated 0)


(protected int numCreated 0)
```

Language Design - methods

```
 public int  get( String key) { return ...; }


 public int  get( String key) { return ...; }
(public      get(        key)  (return x))
(public int  get(        key)  (return x))
(public int  get(        key)          x)
(public int  get((String key))         x)
(public int (get (String key))         x)


(public int (get (String key))         x)
```

Language Design – declare exceptions

```
(public int (get (String key))
    throws NoSuchElementException
           MinusOneException
  (let ((v (get table key)))
    (cond ((== v null)
           (throw (new 'NoSuchElementException)))
          ((< v 0)
           (throw (new 'MinusOneException)))
          (else ...
```

```
(public int (get (String key))
    (throws NoSuchElementException
            MinusOneException)
  (let ((v (get table key)))
    (cond ((== v null)
           (throw (new 'NoSuchElementException)))
          ((< v 0)
           (throw (new 'MinusOneException)))
          (else ...
```

Language Design – declare exceptions

```
(public int (get (String key))
    throws NoSuchElementException
           MinusOneException
  ...)
```

```
(public int (get (String key))
    (throws NoSuchElementException
            MinusOneException)
  ...)
```

Language Design – inheritance

```
public class Point extends PointBase

(public class Point extends PointBase

(public class Point (extends PointBase)
```

Language Design – blocks, statements, expressions

```
public int demo(int x, int y) {
  int tmp1 = x + y;
  ...
  int tmp2 = ...;
  ...
  return tmp2;
}

(public int (demo (int x) (int y))
  (let ((tmp1 (+ x y)))
    ...
    (let ((tmp2 ...))
      ...
      tmp2)))

(public int (demo (int x) (int y))
  (let (= tmp1 (+ x y)))
  ...
  (let (= tmp2 ...))
  ...
  tmp2)

begin    try/catch/finally
all statements are expressions that return values.
```

Language Design – loops and control transfer

last-call-optimization (LCO) to encourage recursion

But deep integration with Java needs to go through Java's non-LCO call stack

loop macros: for **`while`** **`do`**/**`while`**

abrupt transfer of control : **`throw`**

implement **`break`** **`continue`** **`return`**
or provide **`call/ep`**?

Language Design – anonymous classes

```
AccessController.doPrivileged(
  new PrivilegedAction() {
    public Object run() {
      ...
      return null;
    }
  });
```

?

Language Design – namespaces (e.g., Lisp1, Lisp2)

```
public class Namespaces {
    public enum bar { foo };
    public int foo = 3;
    public static void main(String[] x) {
        new Namespaces().demo();
    }
    public void demo() {
        System.out.print(foo + " ");
        System.out.print(foo() + " ");
        int foo = 5;
        System.out.print(foo + " ");
        System.out.print(new foo().foo + " ");
        System.out.print(bar.foo + " ");
    }
    public int foo() { return foo + 1; }
}
class foo {
    public int foo = 6;
}
// prints: 3 4 5 6 foo
```

Outline of Talk

- **Overview of the llava language**
- **llava language design**
- **Brief tour of current implementation**
- **Future work**

implementation – Reflective Invocation

llava's Reflective Invocation (RI) implementation like Skij's

main difference is how llava uses RI

Instead of explicit interface (Skij `invoke`, JScheme's dot-notation) RI is part of llava's variable lookup algorithm:

- llava's undefined identifier handler binds an RI placeholder
- if invoked, the placeholder calls the RI system

plus:

- RI procedure definitions that try the RI system

Implementation – (non) RI procedure definitions

To enable Java classes and llava procedures to coexist, llava provides two forms of procedure definitions

```
(floatValue 10) => 10.0

(define (floatValue x)
  (list 'floatValue x))

(floatValue 's) => (floatValue s)
(floatValue 10) => 10.0

(define floatValue
  (lambda (x)
    (list 'floatValue x)))

(floatValue 's) => (floatValue s)
(floatValue 10) => (floatValue 10)
```

Implementation – central RI technique

```
public class RIProcedure {
    private String name;
    private Lambda defaultLambda;

    public Object apply(Pair args, Engine engine) {
        Object targetObject;
        // Cannot be generic: (foo)  (foo null)
        if (args == null || (targetObject = args.car()) == null)
            return tryDefaultLambdaOrUndefined(args, engine);
        try {
            Object[] methodArgs = List.toArray((Pair)args.cdr());
            Object result =
                RI.invoke(name, targetObject, methodArgs);
            if (result != RI.NoSuchMethod) { return result; }
        } catch (Throwable t) { throw F.newLlavaException(t); }
        return tryDefaultLambdaOrUndefined(args, engine);
    }

    private Object tryDefaultLambdaOrUndefined(Pair args,
                                               Engine engine) {
        if (defaultLambda() != null) {
            return defaultLambda().apply(args, engine);
        } else {
            throw F.newUndefinedIdException(name);
        }
    }
}
```

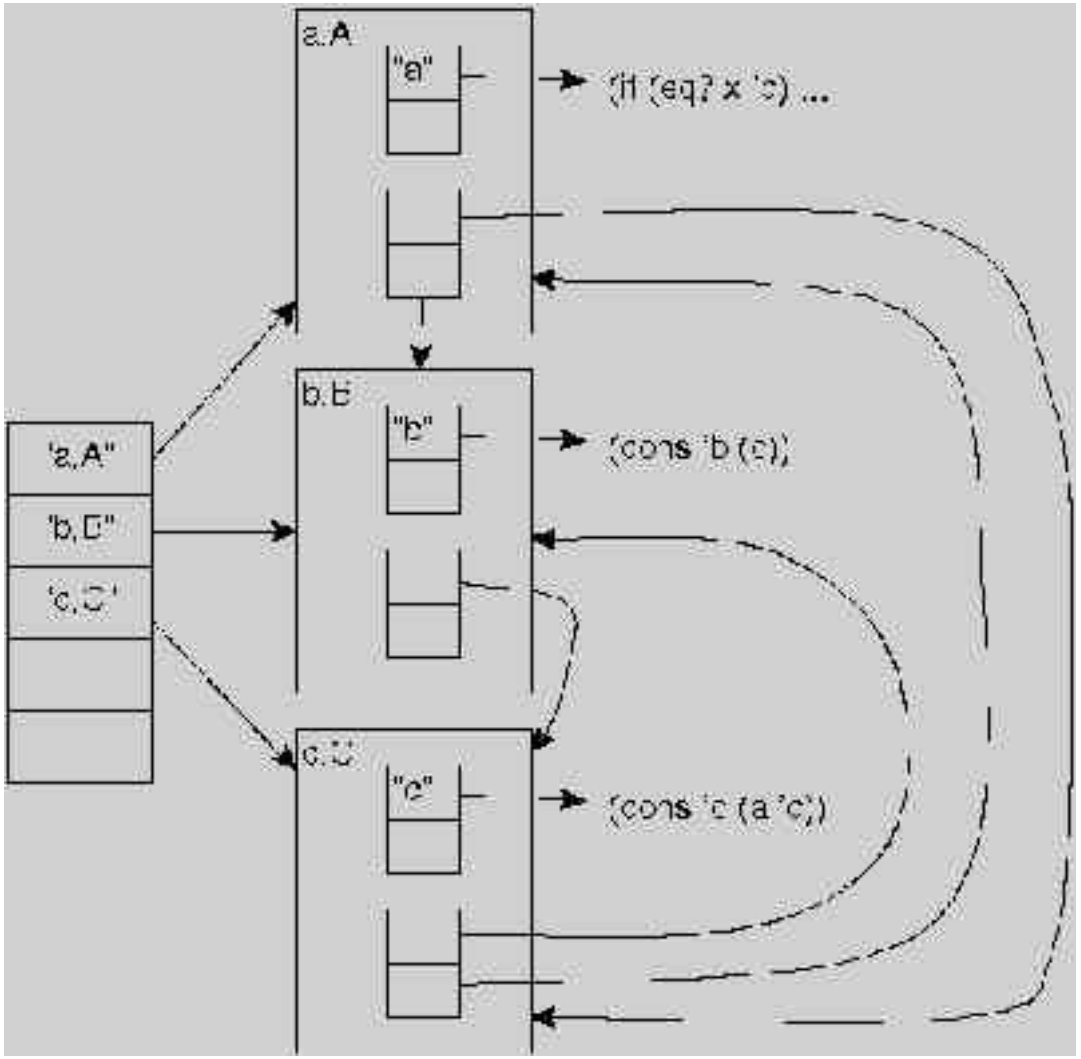
Implementation – package/import

```
(package a.A)
(import c.C)
(import b.B)
(define (a x)
  (if (eq? x 'c)
      'a
      (b)))
```

```
(package b.B)
(import c.C)

(define (b)
  (cons 'b (c)
```

```
(package c.C)
(import b.B)
(import a.A)
(define (c)
  (cons 'c (a 'c)))
```

```
(a 0) => (b c . a)
```



Implementation – package/import requirements

Java classes can be imported.

llava provides a REPL, so packages can be defined on-the-fly interactively and then later become files.

A llava package file of a package already represented internally should not be loaded unless it is a new file or has changed since it was last loaded.

Even if a llava package file has not been touched it is still necessary to search the transitive closure of its import list for any package files that may have changed.

Packages can mutually reference each other so load loops must be detected.

New packages automatically import java.lang.* and org.llava.

Implementation – import algorithm

```
if alreadyImportedInPackage?
    loadLlavaFileIfTouched
else if existsInPackageNameMap
    addToCurrentPackageImportList
    loadLlavaFileIfTouched
else if java class exists
    import class
    classAlreadyImported = true
    addToCurrentPackageImportList
else
    for path in classpath
        if path/file exists
            load file unless being loaded
            addToCurrentPackageImportList
            break
    if file not found in classpath
        throw FileNotFoundException
```

```
loadLlavaFileIfTouched:
if not classAlreadyImported
  for path in classpath
    if path/file exists
      if lastModified = 0 || fileTime > lastModified
        load file
```

Implementation – Engine-based compiler/runtime

```
public class Engine {
    protected Code code;
    public Object run (Code code) {
        this.code = code;
        Object result = code.run(this);
        while (result == this) {
            result = this.code.run(this);
        }
        return result;}
    public Object tailCall (Code code)
        this.code = code;
        return this;}}

public class CodeReference extends Code {
    private int slot;
    public Object run (Engine engine) {
        return frame.get(slot);}}

public class CodeIf extends Code {
    protected Code testCode, thenCode, elseCode;
    public Object run (Engine engine) {
        Boolean test =
            (Boolean) engine.run(testCode);
        if (test.booleanValue() == true) {
            return engine.tailCall(thenCode);
        }
        return engine.tailCall(elseCode);}}
```

Implementation – classes

Java

```
package org.llava.pb;

public abstract class PointBase {
    protected int x = 0;
    public int getX() { return x; }
    protected void move(int dx) {
        x += dx;
    }
    protected abstract String getName();
}
```

llava

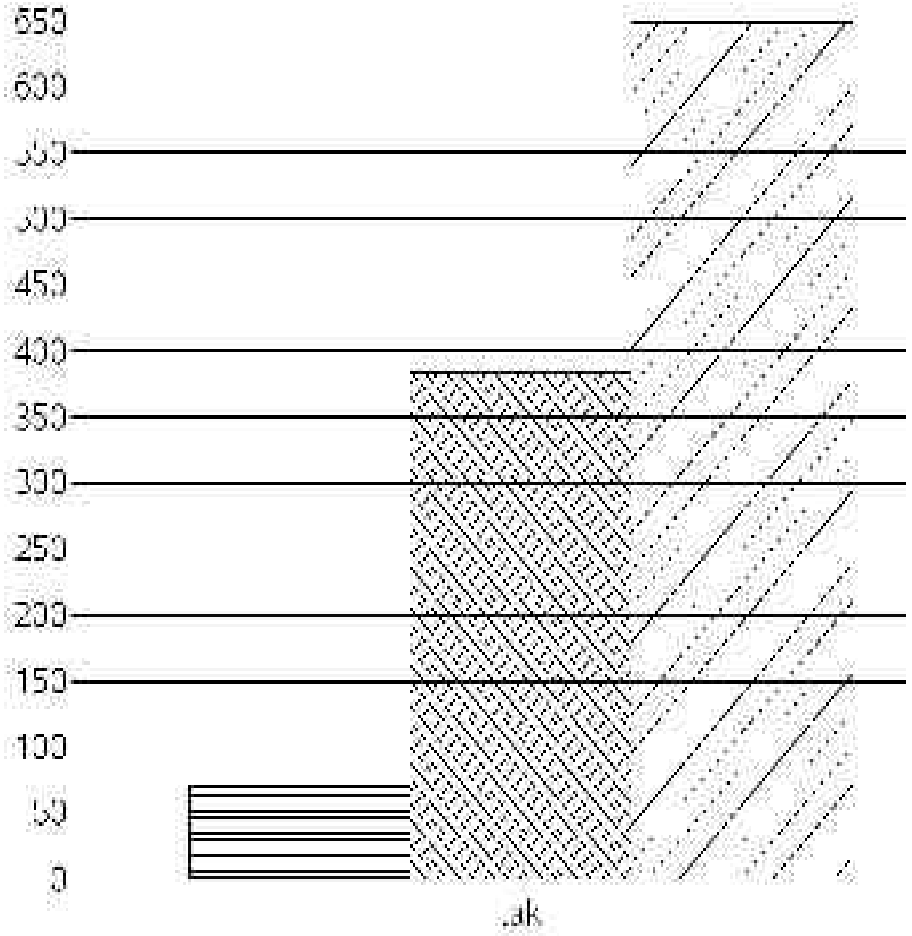
```
(package org.llava.pb)

(public abstract class PointBase
    (protected int x 0)
    (public int (getX) x)
    (protected void (move dx)
        (+= x dx))
    (protected abstract String (getName)))
```

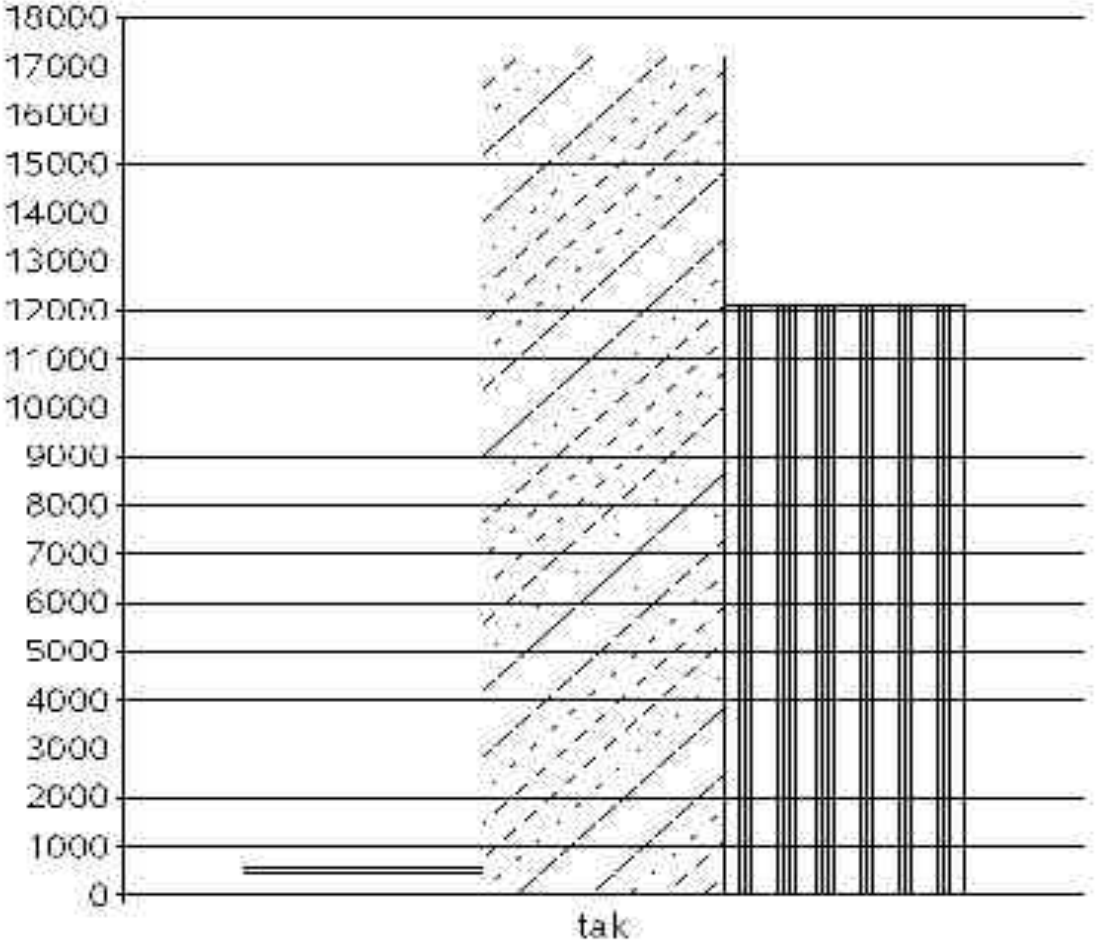
Implementation – compiling classes

```
package org.llava.pb;

import org.llava.F;
import org.llava.Pair;
import org.llava.impl.util.List;

public abstract class PointBase {
    protected int x = 0;
    public int getX() {
        Pair app = List.list(F.newSymbol("PointBase-LLAVA-getX"),
                             this);
        return((Integer)F.ce(app)).intValue();
    }
    protected abstract String getName();
}
```

```
(package org.llava.pb.PointBase-LLAVA)

(define PointBase-LLAVA-getX
  (lambda (this)
    (-f 'x this)))
```

Implementation – performance





Outline of Talk

- **Overview of the llava language**
- **llava language design**
- **Brief tour of current implementation**
- **Future work**

Future Work

- **Renewed focus on llava language design**
- **Redo implementation (from scratch?)**
  - **Avoid “two-worlds” (implies no-LCO?)**
  - **Maximum leverage of Java platform**
    - **IDEs, debuggers, JavaDoc**
- **JSR *, *, ***

Conclusion - llava.org

- llava is Java in List Prefix notation
- Plus macros, procedures, lists, symbols
- Easy for Java programmers to entry Lisp world

```
(let ((bomb 1))
  (define (demo)
    (try
       (if (< bomb 0)
           (throw (new 'Exception "Give up!")))
       (list "Normal result is: " (/ 2 bomb))
     (catch (ArithmeticException e)
       (list "Arithmetic: " e))
     (catch (Exception e)
       (list "Exception: " e))
     (finally
       (set! bomb (- bomb 1))))))
```