Java for Lispers

How to survive in a Java-centric world

António Menezes Leitão

antonio.leitao@dei.ist.utl.pt

IST / INESC-ID

Summary

Problem

Summary

Problem

- Java is everywhere

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything
- Lispers that don't program in Java miss the APIs

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything
- Lispers that don't program in Java miss the APIs
- Lispers that program in Java miss macros, CLOS, lambdas, keyword parameters, loop, format, ...

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything
- Lispers that don't program in Java miss the APIs
- Lispers that program in Java miss macros, CLOS, lambdas, keyword parameters, loop, format, ...

Solution

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything
- Lispers that don't program in Java miss the APIs
- Lispers that program in Java miss macros, CLOS, lambdas, keyword parameters, loop, format, ...

Solution

- Linj is a Common Lisp-like language

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything
- Lispers that don't program in Java miss the APIs
- Lispers that program in Java miss macros, CLOS, lambdas, keyword parameters, loop, format, ...

Solution

- Linj is a Common Lisp-like language
- The Linj compiler translates from Linj to **readable** Java

Summary

Problem

- Java is everywhere
- Java is a frequent requirement in industry projects
- There are Java APIs for everything
- Lispers that don't program in Java miss the APIs
- Lispers that program in Java miss macros, CLOS, lambdas, keyword parameters, loop, format, ...

Solution

- Linj is a Common Lisp-like language
- The Linj compiler translates from Linj to **readable** Java
- The Jnil compiler translates from Java to **readable** Linj

Outline

- Linj
- Jnil
- Conclusions

A Linj Program

A Linj Program

```
(defun fact (n)
  (if (= n 0)
    1
    (* n (fact (1- n)))))
```

A Linj Program

```
(defun fact (n)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

Compile & Run & Run & . . .

```
(defun fact (n)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

Compile & Run & Run & . . .

```
(defun fact (n)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))

(fact 0) -> 1
(fact 10) -> 3628800
(fact 20) -> 2432902008176640000
(fact 30) -> 265252859812191058636308480000000
(fact 40) -> 815915283247897734345611269596115894272000000000
(fact 50) -> 30414093201713378043612608166064768844377641568960512O...
```

How?

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {




}
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {




}
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {




}
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {

    } else {

    }
}
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {

    }
}
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
```

How?

```
(defun fact (n)
  (if (= n 0)
      1
      (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact((Bignum)n));
}
```

Final steps

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact((Bignum)n));
}
```

Final steps

```
public class Fact extends Object {

    public static Bignum fact(Bignum n) {
        if (n.compareTo(Bignum.valueOf(0)) == 0) {
            return Bignum.valueOf(1);
        } else {
            return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
        }
    }

    public static void main(String[] args) {
        Object n = new LinjReader(new StringReader(args[0]), "").read(null);
        System.out.print("(fact ");
        System.out.print(n);
        System.out.print(") -> ");
        System.out.println(fact((Bignum)n));
    }
}
```

Final steps

```
import java.io.StringReader;
import linj.Bignum;
import linj.LinjReader;

public class Fact extends Object {

    public static Bignum fact(Bignum n) {
        if (n.compareTo(Bignum.valueOf(0)) == 0) {
            return Bignum.valueOf(1);
        } else {
            return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
        }
    }

    public static void main(String[] args) {
        Object n = new LinjReader(new StringReader(args[0]), "").read(null);
        System.out.print("(fact ");
        System.out.print(n);
        System.out.print(") -> ");
        System.out.println(fact((Bignum)n));
    }
}
```

Declaring Types

```
(defun fact (n)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact((Bignum)n));
}
```

Declaring Types

```
(defun fact (n) (declare (int n))
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static Bignum fact(Bignum n) {
    if (n.compareTo(Bignum.valueOf(0)) == 0) {
        return Bignum.valueOf(1);
    } else {
        return n.multiply(fact(n.subtract(Bignum.valueOf(1))));
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact((Bignum)n));
}
```

Declaring Types

```
(defun fact (n) (declare (int n))
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact(((Number)n).intValue()));
}
```

Declaring Types

```
(defun fact (n/int)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact(((Number)n).intValue()));
}
```

Declaring Types

```
(defun fact (n/int)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n/int)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}

public static void main(String[] args) {
    Object n = new LinjReader(new StringReader(args[0]), "").read(null);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact(((Number)n).intValue()));
}
```

Declaring Types

```
(defun fact (n/int)
  (if (= n 0)
    1
    (* n (fact (1- n)))))

(defun main (n/int)
  (format t "(fact ~A) -> ~A~%" n (fact n)))
```

```
public static int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    System.out.print("(fact ");
    System.out.print(n);
    System.out.print(") -> ");
    System.out.println(fact(n));
}
```

Classes & Methods

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {









}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {









}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {









    protected int x;
    protected int y;
}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {









    protected int x;
    protected int y;
}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {




    public int shapeX() { return x; }

    public int shapeY() { return y; }




    protected int x;
    protected int y;
}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {




    public int shapeX() { return x; }

    public int shapeY() { return y; }




    protected int x;
    protected int y;
}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {

    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int shapeX() { return x; }

    public int shapeY() { return y; }









    protected int x;
    protected int y;
}
```

Classes & Methods

```
(defclass shape ()
  ((x :type int :reader shape-x :initarg :x)
   (y :type int :reader shape-y :initarg :y)))

(defmethod move-to ((figure shape) new-x new-y)
  (with-slots (x y) figure
    (setf x new-x
          y new-y)))
```

```
public class Shape extends Object {

    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int shapeX() { return x; }

    public int shapeY() { return y; }

    public void moveTo(int newX, int newY) {
        x = newX;
        y = newY;
    }

    protected int x;
    protected int y;
}
```

Classes & Methods

```
(defclass rectangle (shape)
  ((width :type int :accessor rectangle-width :initarg :width)
   (height :type int :accessor rectangle-height :initarg :height)))
```

Classes & Methods

```
(defclass rectangle (shape)
  ((width :type int :accessor rectangle-width :initarg :width)
   (height :type int :accessor rectangle-height :initarg :height)))

(defclass circle (shape)
  ((radius :type int :accessor circle-radius :initarg :radius)))
```

Classes & Methods

```
(defclass rectangle (shape)
  ((width :type int :accessor rectangle-width :initarg :width)
   (height :type int :accessor rectangle-height :initarg :height)))

(defclass circle (shape)
  ((radius :type int :accessor circle-radius :initarg :radius)))

(defmethod draw ((figure shape) g/graphics)
  (error "Method draw not implemented on ~A" figure))
```

Classes & Methods

```
(defclass rectangle (shape)
  ((width :type int :accessor rectangle-width :initarg :width)
   (height :type int :accessor rectangle-height :initarg :height)))

(defclass circle (shape)
  ((radius :type int :accessor circle-radius :initarg :radius)))

(defmethod draw ((figure shape) g/graphics)
  (error "Method draw not implemented on ~A" figure))

(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g
             (shape-x figure)
             (shape-y figure)
             (rectangle-width figure)
             (rectangle-height figure)))
```

Classes & Methods

```
(defclass rectangle (shape)
  ((width :type int :accessor rectangle-width :initarg :width)
   (height :type int :accessor rectangle-height :initarg :height)))

(defclass circle (shape)
  ((radius :type int :accessor circle-radius :initarg :radius)))

(defmethod draw ((figure shape) g/graphics)
  (error "Method draw not implemented on ~A" figure))

(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g
             (shape-x figure)
             (shape-y figure)
             (rectangle-width figure)
             (rectangle-height figure)))

(defmethod draw ((figure circle) g/graphics)
  (let ((diameter (* 2 (circle-radius figure))))
    (fill-oval g
               (shape-x figure)
               (shape-y figure)
               diameter
               diameter)))
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)
```

```
new Rectangle(30, 0, 10, 20);
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)

(defun member (elem list &key (test #'eq) (key #'identity))
  (loop for l on list
        until (funcall test elem (funcall key (first l)))
        finally return l))
```

```
new Rectangle(30, 0, 10, 20);
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)

(defun member (elem list &key (test #'eq) (key #'identity))
  (loop for l on list
        until (funcall test elem (funcall key (first l)))
        finally return l))
```

```
new Rectangle(30, 0, 10, 20);

public static Cons member(Object elem, Cons list, Predicate2 test, Function key) {
    Cons l = list;
    for (; ! l.endp(); l = l.rest()) {
        if (test.funcall(elem, key.funcall(l.first()))) {
            break;
        }
    }
    return l;
}
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)

(defun member (elem list &key (test #'eq) (key #'identity))
  (loop for l on list
        until (funcall test elem (funcall key (first l)))
        finally return l))
```

```
new Rectangle(30, 0, 10, 20);

public static Cons member(Object elem, Cons list, Predicate2 test, Function key) {
    Cons l = list;
    for (; ! l.endp(); l = l.rest()) {
        if (test.funcall(elem, key.funcall(l.first()))) {
            break;
        }
    }
    return l;
}

public static Cons memberKey(..., int argsPassed) {
    return member(elem, list,
                  ((argsPassed & 1) == 0) ? Predicate2.EQ_FUNCTION : test,
                  ((argsPassed & 2) == 0) ? Cons.IDENTITY_FUNCTION : key);
}
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)

(defun member (elem list &key (test #'eq) (key #'identity))
  (loop for l on list
        until (funcall test elem (funcall key (first l)))
        finally return l))

(member e lst :test #'eql)
```

```
new Rectangle(30, 0, 10, 20);

public static Cons member(Object elem, Cons list, Predicate2 test, Function key) {
    Cons l = list;
    for (; ! l.endp(); l = l.rest()) {
        if (test.funcall(elem, key.funcall(l.first()))) {
            break;
        }
    }
    return l;
}

public static Cons memberKey(..., int argsPassed) {
    return member(elem, list,
                  ((argsPassed & 1) == 0) ? Predicate2.EQ_FUNCTION : test,
                  ((argsPassed & 2) == 0) ? Cons.IDENTITY_FUNCTION : key);
}

memberKey(e, lst, Predicate2.EQL_FUNCTION, null, 1)
```

Optional and Keyword parameters

```
(make-instance 'rectangle :x 10 :width 30 :y 20)

(defun member (elem list &key (test #'eq) (key #'identity))
  (loop for l on list
        until (funcall test elem (funcall key (first l)))
        finally return l))

(member e lst :test #'eql)
(member e lst :test #'eql :key #'first)
```

```
new Rectangle(30, 0, 10, 20);

public static Cons member(Object elem, Cons list, Predicate2 test, Function key) {
    Cons l = list;
    for (; ! l.endp(); l = l.rest()) {
        if (test.funcall(elem, key.funcall(l.first()))) {
            break;
        }
    }
    return l;
}

public static Cons memberKey(..., int argsPassed) {
    return member(elem, list,
                  ((argsPassed & 1) == 0) ? Predicate2.EQ_FUNCTION : test,
                  ((argsPassed & 2) == 0) ? Cons.IDENTITY_FUNCTION : key);
}

memberKey(e, lst, Predicate2.EQL_FUNCTION, null, 1)
member(e, lst, Predicate2.EQL_FUNCTION, Cons.CAR_FUNCTION)
```

Mixins & Method Combination

Mixins & Method Combination

```
(defmixin colored-mixin ()
  ((color :type color :initarg :color :reader get-color :writer set-color)))
```

Mixins & Method Combination

```
(defmixin colored-mixin ()
  ((color :type color :initarg :color :reader get-color :writer set-color)))

(defmethod draw :around ((figure colored-mixin) g/graphics)
  (let ((prev-color (get-color g)))
    (set-color g (get-color figure))
    (unwind-protect
        (call-next-method)
      (set-color g prev-color))))
```

Mixins & Method Combination

```
(defmixin colored-mixin ()
  ((color :type color :initarg :color :reader get-color :writer set-color)))

(defmethod draw :around ((figure colored-mixin) g/graphics)
  (let ((prev-color (get-color g)))
    (set-color g (get-color figure))
    (unwind-protect
        (call-next-method)
      (set-color g prev-color))))
```

```
interface ColoredMixin {

    public abstract Color getColor();

    public abstract Color setColor(Color color);
}
```

Mixins & Method Combination

```
(defmixin colored-mixin ()
  ((color :type color :initarg :color :reader get-color :writer set-color)))

(defmethod draw :around ((figure colored-mixin) g/graphics)
  (let ((prev-color (get-color g)))
    (set-color g (get-color figure))
    (unwind-protect
        (call-next-method)
      (set-color g prev-color))))

(defclass colored-rectangle (colored-mixin rectangle)
  ())

(defclass colored-circle (colored-mixin circle)
  ())
```

```
interface ColoredMixin {

    public abstract Color getColor();

    public abstract Color setColor(Color color);
}
```

Mixins & Method Combination

```
class ColoredRectangle extends Rectangle implements ColoredMixin {









}
```

Mixins & Method Combination

```
class ColoredRectangle extends Rectangle implements ColoredMixin {

    public ColoredRectangle(Color color, int width, int height, int x, int y) {
        super(width, height, x, y);
        this.color = color;
    }

    public Color getColor() {
        return color;
    }

    public Color setColor(Color color) {
        return this.color = color;
    }









    protected Color color;
}
```

Mixins & Method Combination

```
class ColoredRectangle extends Rectangle implements ColoredMixin {

    public ColoredRectangle(Color color, int width, int height, int x, int y) {
        super(width, height, x, y);
        this.color = color;
    }

    public Color getColor() {
        return color;
    }

    public Color setColor(Color color) {
        return this.color = color;
    }

    public void draw(Graphics g) {
        Color prevColor = g.getColor();
        g.setColor(getColor());
        try {
            primaryDraw(g);
        } finally {
            g.setColor(prevColor);
        }
    }




    protected Color color;
}
```

Mixins & Method Combination

```
class ColoredRectangle extends Rectangle implements ColoredMixin {

    public ColoredRectangle(Color color, int width, int height, int x, int y) {
        super(width, height, x, y);
        this.color = color;
    }

    public Color getColor() {
        return color;
    }

    public Color setColor(Color color) {
        return this.color = color;
    }

    public void draw(Graphics g) {
        Color prevColor = g.getColor();
        g.setColor(getColor());
        try {
            primaryDraw(g);
        } finally {
            g.setColor(prevColor);
        }
    }

    public void primaryDraw(Graphics g) {
        super.draw(g);
    }

    protected Color color;
}
```

Mixins & Method Combination

```
(defmixin colored-mixin () ...)

(defmethod draw :around ((figure colored-mixin) g/graphics) ...)

(defclass colored-rectangle (colored-mixin rectangle)
  ())
```

Mixins & Method Combination

```
(defmixin colored-mixin () ...)

(defmethod draw :around ((figure colored-mixin) g/graphics) ...)

(defclass colored-rectangle (colored-mixin rectangle)
  ())

(defclass outlined-rectangle (colored-rectangle)
  ())
```

Mixins & Method Combination

```
(defmixin colored-mixin () ...)

(defmethod draw :around ((figure colored-mixin) g/graphics) ...)

(defclass colored-rectangle (colored-mixin rectangle)
  ())

(defclass outlined-rectangle (colored-rectangle)
  ())

(defmethod draw ((figure outlined-rectangle) g/graphics)
  (draw-rect g
             (shape-x figure)
             (shape-y figure)
             (rectangle-width figure)
             (rectangle-height figure)))
```

Mixins & Method Combination

```
(defmixin colored-mixin () ...)

(defmethod draw :around ((figure colored-mixin) g/graphics) ...)

(defclass colored-rectangle (colored-mixin rectangle)
  ())

(defclass outlined-rectangle (colored-rectangle)
  ())

(defmethod draw ((figure outlined-rectangle) g/graphics)
  (draw-rect g
             (shape-x figure)
             (shape-y figure)
             (rectangle-width figure)
             (rectangle-height figure)))
```

```
class OutlinedRectangle extends ColoredRectangle {

    public OutlinedRectangle(Color color, int width, int height, int x, int y) {
        super(color, width, height, x, y);
    }



}
```

Mixins & Method Combination

```
(defmixin colored-mixin () ...)

(defmethod draw :around ((figure colored-mixin) g/graphics) ...)

(defclass colored-rectangle (colored-mixin rectangle)
  ())

(defclass outlined-rectangle (colored-rectangle)
  ())

(defmethod draw ((figure outlined-rectangle) g/graphics)
  (draw-rect g
             (shape-x figure)
             (shape-y figure)
             (rectangle-width figure)
             (rectangle-height figure)))
```

```
class OutlinedRectangle extends ColoredRectangle {

    public OutlinedRectangle(Color color, int width, int height, int x, int y) {
        super(color, width, height, x, y);
    }

    public void primaryDraw(Graphics g) {
        g.drawRect(shapeX(), shapeY(), rectangleWidth(), rectangleHeight());
    }
}
```

Extending Java Classes

Extending Java Classes

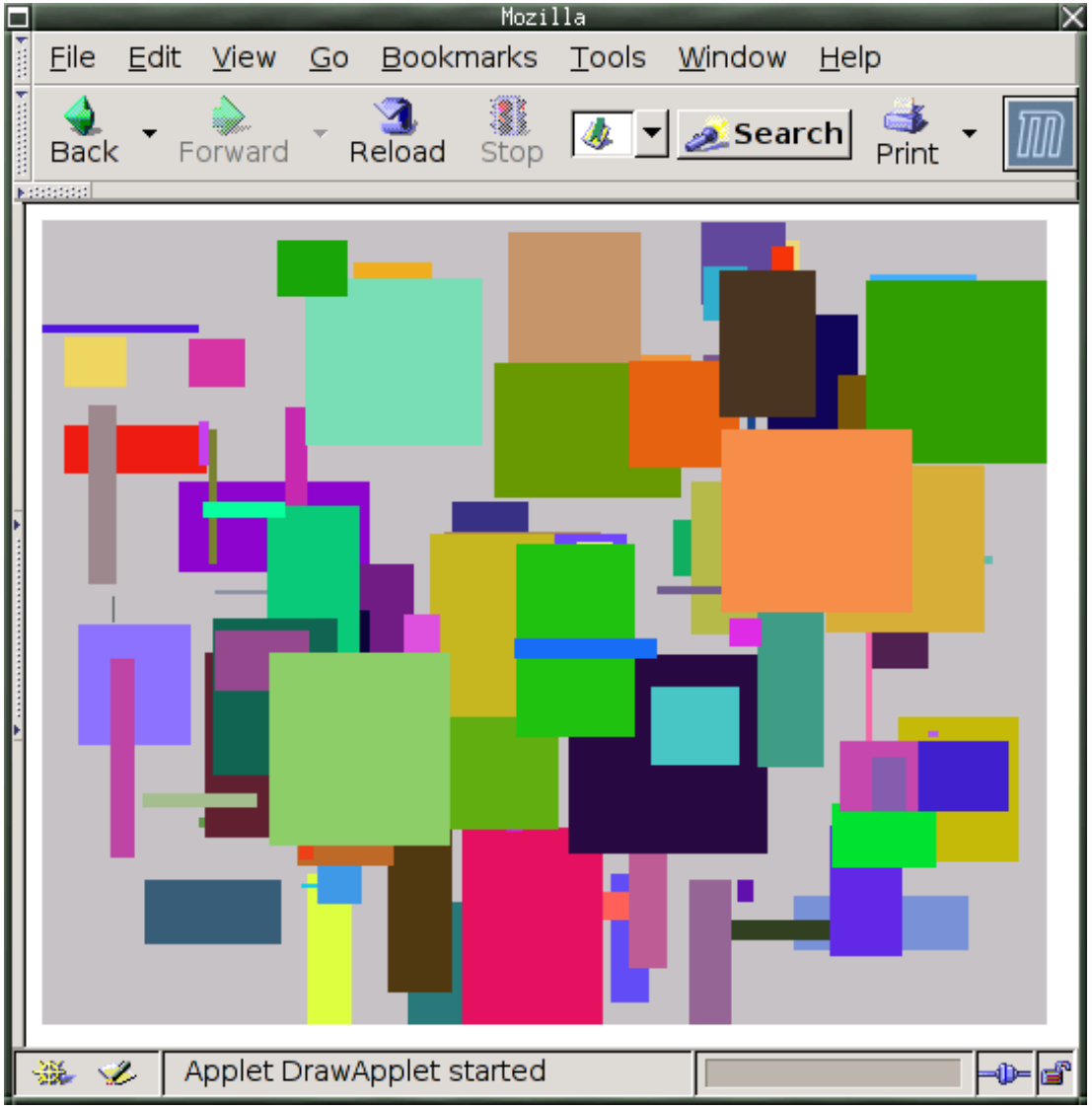
```
(defclass draw-applet (applet)
  ())

(defmethod paint :after ((applet draw-applet) g/graphics)
  (let ((rect (make-instance 'colored-rectangle)))
    (dotimes (i (random 100))
      (move-to rect (random 450) (random 350))
      (resize-to rect (random 100) (random 100))
      (let ((color (new 'color (random 256) (random 256) (random 256))))
        (set-color color rect)
        (draw rect g)))))
```

Extending Java Classes

```
(defclass draw-applet (applet)
  ())

(defmethod paint :after ((applet draw-applet) g/graphics)
  (let ((rect (make-instance 'colored-rectangle)))
    (dotimes (i (random 100))
      (move-to rect (random 450) (random 350))
      (resize-to rect (random 100) (random 100))
      (let ((color (new 'color (random 256) (random 256) (random 256))))
        (set-color color rect)
        (draw rect g)))))
```

```
public class DrawApplet extends Applet {

    public void paint(Graphics g) {
        primaryPaint(g);
        afterPaintDrawApplet(g);
    }

    public void afterPaintDrawApplet(Graphics g) {
        ColoredRectangle rect = new ColoredRectangle(null, 0, 0, 0, 0, 0);
        int limit = (int)(Math.random() * 100);
        for (int i = 0; i < limit; ++i) {
            rect.moveTo(...);
            ...
            rect.draw(g);
        }
    }
}
```

The Result


Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop Search Print
Applet DrawApplet started

Multiple Dispatch Methods

```
(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure)))
```

Multiple Dispatch Methods

```
(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure)))

(defmethod draw ((figure rectangle) (g graphics-2-d))
  (in (the alpha-composite)
    (let ((old-composite (get-composite g)))
      (set-composite g (get-instance SRC_OVER 0.4))
      (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure))
      (set-composite g old-composite))))
```

Multiple Dispatch Methods

```
(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure)))

(defmethod draw ((figure rectangle) (g graphics-2-d))
  (in (the alpha-composite)
    (let ((old-composite (get-composite g)))
      (set-composite g (get-instance SRC_OVER 0.4))
      (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure))
      (set-composite g old-composite))))
```

```
public void draw(Graphics g) {



       g.fillRect(shapeX(), shapeY(), rectangleWidth(), rectangleHeight());

}
```

Multiple Dispatch Methods

```
(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure)))

(defmethod draw ((figure rectangle) (g graphics-2-d))
  (in (the alpha-composite)
    (let ((old-composite (get-composite g)))
      (set-composite g (get-instance SRC_OVER 0.4))
      (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure))
      (set-composite g old-composite))))
```

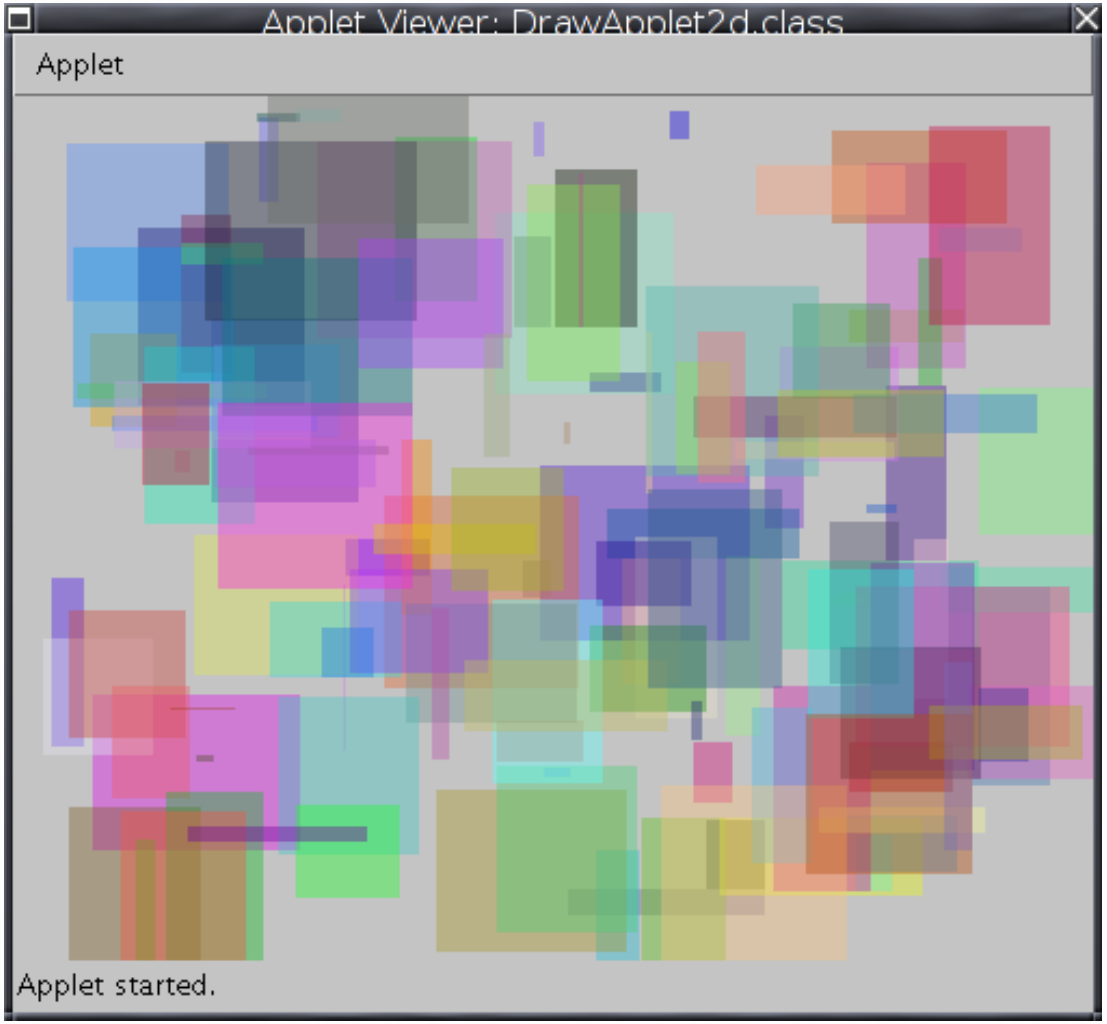
```
public void draw(Graphics g) {



        g.fillRect(shapeX(), shapeY(), rectangleWidth(), rectangleHeight());

}

public void draw(Graphics2D g) {
    Composite oldComposite = g.getComposite();
    g.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.4f));
    g.fillRect(shapeX(), shapeY(), rectangleWidth(), rectangleHeight());
    g.setComposite(oldComposite);
}
```

Multiple Dispatch Methods

```
(defmethod draw ((figure rectangle) g/graphics)
  (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure)))

(defmethod draw ((figure rectangle) (g graphics-2-d))
  (in (the alpha-composite)
    (let ((old-composite (get-composite g)))
      (set-composite g (get-instance SRC_OVER 0.4))
      (fill-rect g (shape-x figure) (shape-y figure) (rectangle-width figure) (rectangle-height figure))
      (set-composite g old-composite))))
```

```
public void draw(Graphics g) {
    if ((g == null) || (g instanceof Graphics2D)) {
        draw((Graphics2D)g);
    } else {
        g.fillRect(shapeX(), shapeY(), rectangleWidth(), rectangleHeight());
    }
}

public void draw(Graphics2D g) {
    Composite oldComposite = g.getComposite();
    g.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.4f));
    g.fillRect(shapeX(), shapeY(), rectangleWidth(), rectangleHeight());
    g.setComposite(oldComposite);
}
```

The Result



Other Linj features

Linj supports several Common Lisp features, such as

- Data types: bignums, rationals, conses, symbols, hashtables, vectors, etc.
- Optional, keyword and rest parameters
- Multiple values
- Restricted lambda expressions
- Macros
- Trace and profile
- Futures
- Perfect integration with Java APIs

Problems

Problems

- What happens if the generated Java code is modified by someone else?

Problems

- What happens if the generated Java code is modified by someone else?
- Can I use the Java APIs without using Linj?

Problems

- What happens if the generated Java code is modified by someone else?
- Can I use the Java APIs without using Linj?

Solutions

Problems

- What happens if the generated Java code is modified by someone else?
- Can I use the Java APIs without using Linj?

Solutions

- There is jlinker, jfli, jacol, . . .

Problems

- What happens if the generated Java code is modified by someone else?
- Can I use the Java APIs without using Linj?

Solutions

- There is jlinker, jfli, jacol, . . .
- It is possible (but tedious) to translate from Java to Common Lisp

Problems

- What happens if the generated Java code is modified by someone else?
- Can I use the Java APIs without using Linj?

Solutions

- There is jlinker, jfli, jacol, . . .
- It is possible (but tedious) to translate from Java to Common Lisp
- Jnil (Tiago Maduro-Dias and Rui Curto) automates that translation (but the target is Linj)

Problems

- What happens if the generated Java code is modified by someone else?
- Can I use the Java APIs without using Linj?

Solutions

- There is jlinker, jfli, jacol, . . .
- It is possible (but tedious) to translate from Java to Common Lisp
- Jnil (Tiago Maduro-Dias and Rui Curto) automates that translation (but the target is Linj)
- It is possible (and not tedious) to translate from Linj to Common Lisp

Jnil - From Java to Linj

First major milestone: Joda-Time

- A replacement for JDK Calendar
- Date and time, date without time, and time without date
- Instants, intervals, time periods, time durations
- Multiple calendar systems and the full range of time zones
- 177 classes, 23 interfaces, 2500 methods
- Almost all Joda-Time files were translated into Linj
- Still more work to do but looks very promising

From Java to Linj

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this))))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))









          (post-incf i))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))









           (post-incf i))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
            (let* ((index/int (index-of this field-type)))



           (post-incf i))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
              (cond ((>= index 0)




          (post-incf i))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                            (add-wrap-partial (get-field this index) this index new-values
                                              (in (the field-utils)
                                                  (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
```

From Java to Linj

```java
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```lisp
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                            (add-wrap-partial (get-field this index) this index new-values
                                              (in (the field-utils)
                                                  (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

From Java to Linj and back to Java

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

From Java to Linj and back to Java

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) throws CloneNotSupportedException {
    if ((period == null) || (scalar == 0)) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; true; ) {
        if (i >= period.size()) { break; }
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
        i++;
    }
    return new TimeOfDay(this, newValues);
}
```

From Java to Linj and back to Java

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) throws CloneNotSupportedException {
    if ((period == null) || (scalar == 0)) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; true; ) {
        if (i >= period.size()) { break; }
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
        i++;
    }
    return new TimeOfDay(this, newValues);
}
```

From Java to Linj and back to Java

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) throws CloneNotSupportedException {
    if ((period == null) || (scalar == 0)) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; true; ) {
        if (i >= period.size()) { break; }
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
        i++;
    }
    return new TimeOfDay(this, newValues);
}
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
    (return this))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
    (return this))
  (let ((new-values (get-values this)))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
          do (let* ((field-type/duration-field-type (get-field-type period i)))
               (let* ((index/int (index-of this field-type)))
                 (cond ((>= index 0)
                        (setf new-values
                         (add-wrap-partial (get-field this index) this index new-values
                                           (in (the field-utils)
                                               (safe-multiply-to-int (get-value period i) scalar))))))))
          (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
    (return this))
  (let ((new-values (get-values this)))
    (dotimes (i (size period))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
    (return this))
  (let ((new-values (get-values this)))
    (dotimes (i (size period))
      (let* ((field-type (get-field-type period i))
             (index (index-of this field-type)))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
    (return this))
  (let ((new-values (get-values this)))
    (dotimes (i (size period))
      (let* ((field-type (get-field-type period i))
             (index (index-of this field-type)))
        (when (>= index 0)
          (setf new-values
                (add-wrap-partial (get-field this index) this index new-values
                                  (in (the field-utils)
                                      (safe-multiply-to-int (get-value period i) scalar)))))))))
```

Refactoring

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (declare (returns time-of-day))
  (cond ((or (eq period null) (= scalar 0))
         (return-from with-period-added this)))
  (let* ((new-values/int[] (get-values this)))
    (loop with i/int = 0 while (< i (size period))
        do (let* ((field-type/duration-field-type (get-field-type period i)))
             (let* ((index/int (index-of this field-type)))
               (cond ((>= index 0)
                      (setf new-values
                       (add-wrap-partial (get-field this index) this index new-values
                                         (in (the field-utils)
                                             (safe-multiply-to-int (get-value period i) scalar))))))))
           (post-incf i))
    (return-from with-period-added (new 'time-of-day this new-values))))
```

```
(defmethod with-period-added ((this time-of-day) period/readable-period scalar/int)
  (when (or (eq period null) (= scalar 0))
    (return this))
  (let ((new-values (get-values this)))
    (dotimes (i (size period))
      (let* ((field-type (get-field-type period i))
             (index (index-of this field-type)))
        (when (>= index 0)
          (setf new-values
                (add-wrap-partial (get-field this index) this index new-values
                                  (in (the field-utils)
                                      (safe-multiply-to-int (get-value period i) scalar)))))))
    (new 'time-of-day this new-values)))
```

Original and Translation

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) throws CloneNotSupportedException {
    if ((period == null) || (scalar == 0)) {
        return this;
    }
    int[] newValues = getValues();
    int limit = period.size();
    for (int i = 0; i < limit; ++i) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

Original and Translation

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) {
    if (period == null || scalar == 0) {
        return this;
    }
    int[] newValues = getValues();
    for (int i = 0; i < period.size(); i++) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

```
public TimeOfDay withPeriodAdded(ReadablePeriod period, int scalar) throws CloneNotSupportedException {
    if ((period == null) || (scalar == 0)) {
        return this;
    }
    int[] newValues = getValues();
    int limit = period.size();
    for (int i = 0; i < limit; ++i) {
        DurationFieldType fieldType = period.getFieldType(i);
        int index = indexOf(fieldType);
        if (index >= 0) {
            newValues = getField(index).addWrapPartial(this, index, newValues,
                    FieldUtils.safeMultiplyToInt(period.getValue(i), scalar));
        }
    }
    return new TimeOfDay(this, newValues);
}
```

Conclusions

- Linj translates from a Common Lisp-like language into Java
- Linj allows us to write programs that can use Java's APIs
- Linj has many of the Common Lisp features that substantially reduce the programming effort
- Jnil translates from Java into Linj
- We are working on the translation from Linj to Common Lisp
- Questions?