# A statically safe alternative

# to virtual types

Part II: A proposal

# Extending Java

Based on earlier proposal extending Java:

- Emulating features of LOOM:
  - *ThisType* as interface of *this*
  - Exact types -- needed for binary methods

```
public interface ListIfc {
  public char head ();
  public @ThisType tail ();
  public void setHead (char h);
  public void setTail (@ThisType t);
}

public interface LenListIfc extends ListIfc {
  int length();
}
```

```
public class List implements @ListIfc {
  protected char h;
  protected @ThisType t;
  public List (char h, @ThisType t) {
    super(); setHead(h); setTail(t);  }
  public char head () { return h; }
  public @ThisType tail () { return t; }
  public void setHead (char h) { this.h=h; }
  public void setTail (@ThisType t) { this.t=t; }
}

public class LenList extends List
  implements @LenListIfc {
  protected int l;
  public LenList(char h, @ThisType t) {super(h,t);}
  public void setTail (@ThisType t) {
    super.setTail(t);
    if (t == null) l = 1; else l = 1+t.length();}
  public int length() { return l; }
}
```

# Generalize Naming of ThisType

```
public interface ListIfc (TType) {
  public char head ();
  public @TType tail ();
  public void setHead (char h);
  public void setTail (@TType t);
}

public class List (TType) implements @ListIfc {
  protected char h;
  protected @TType t;
  public List (char h, @TType t) {
    super(); setHead(h); setTail(t);
  }
  public char head () { return h; }
  public @TType tail () { return t; }
  public void setHead (char h) { this.h=h; }
  public void setTail (@TType t) { this.t=t; }
}
```

# Inner Interfaces Package Recursion

```
public interface AltListGrpIfc {

  public interface XListIfc (XThis) {
    char head ();
    @YThis tail ();
    void setHead (char h);
    void setTail (@YThis t);
  }

  public interface YListIfc (YThis) {
    float head ();
    @XThis tail ();
    void setHead (float h);
    void setTail (@XThis t);
  }
}
```

# Inner Classes for Implementations

```
public class AltListGrp implements AltListGrpIfc {

  public static class XList (XThis)
                              implements @XListIfc{
    protected char h;
    protected @YThis t;
    public XList (char h, @YThis t) {
      super(); setHead(h); setTail(t); }
    public char head () { return h; }
    public @YThis tail () { return t; }
    public void setHead (char h) { this.h=h; }
    public void setTail (@YThis t) { this.t=t; }
  }

  public static class YList (YThis)
    implements @YListIfc
  { ... @XThis }
}
```

# What about type checking?

```
public static class XList (XThis)
                          implements @XListIfc {...}

public static class YList (YThis)
                          implements @YListIfc {...}
```

Interface of `this` in **`XList`** is **`@XThis`**

Interface of `this` in **`YList`** is **`@YThis`**

In both classes assume

**`XThis`** extends **`XListIfc`**

**`YThis`** extends **`YListIfc`**

When extend classes in subclass of *outer* class, assumptions remain true.

# Defining Extensions: Interfaces

```
public interface LenAltListGrpIfc
                              extends AltListGrpIfc{

  public interface LenXListIfc (XThis)
                                  extends XListIfc
  {
    public int length();
  }
  public interface LenYListIfc (YThis)
                                  extends YListIfc
  {
    public int length();
  }
}
```

# Defining Extensions: Classes

```
public class LenAltListGp extends AltListGp{

  public static class LenXList (XThis)
            extends XList implements @LenXListIfc
  {
    protected int l;
    public LenXList(char h, @YThis t) {super(h,t);}
    public void setTail (@YThis t) {
      super.setTail(t);
      if (t == null) l = 1; else l = 1+t.length();}
    public int length () { return l; }
  }

  public static class LenYList (YThis)
            extends YList implements @LenYListIfc
  {...}
}
```

# Using Recursive Interfaces & Classes

```
public class Useit {
  public void useit () {
    @AltListGrpIfc.XThis xl
                = new AltListGrp.XList(`a',null);
    @AltListGrpIfc.YThis yl
                = new AltListGrp.YList(`b',null);
    xl.setTail(yl);
  }
}
```

# Polymorphism

```
public interface PolyAltListGrpIfc
  <XType extends Object, YType extends Object>
{
  public interface XListIfc (XThis) {
    public XType head ();
    public @YThis tail ();
    public void setHead (XType h);
    public void setTail (@YThis t);}

  public interface YListIfc (YThis) {
    public YType head ();
    public @XThis tail ();
    public void setHead (YType h);
    public void setTail (@XThis t); }
}

public class Useit<T extends AltListGrpIfc> {
  ... T.XThis ... T.YThis ...
```

# Summary

- Virtual types and parametric polymorphism have different strengths.
- Parametric polymorphism can be statically typed.
- Virtual types originally required dynamic typing.
- Our proposal gives expressiveness of virtual types but statically safe.
- Inner interfaces / classes support grouping of

  mutually recursive interfaces / classes.
- Generalization of "`ThisType`" construct.
- Fits in well with parametric polymorphism.