LOOJ: Weaving LOOM into Java

Nate Foster
University of Pennsylvania

joint work with
Kim Bruce
Williams College

Introduction

- Java v1.5 has bounded polymorphism!
- But still difficult to express binary methods naturally.
  - A method is a **binary method** if it is intended to be used with a parameter of the same type as the object it is called from.
  - Tricky to write methods with this property in languages with inheritance.
- *MyType* was key feature of TOOPLE, PolyTOIL, LOOM.
  - Self-reflexive type.
- Goal: add *MyType* to Java.

Introduction

- More precise goal: *seamlessly* integrate *MyType* with other Java features including bounded polymorphism and interfaces.
- Key challenge:
  - In Java, objects are described by both class types and interface types in the static type system.
  - Interacts with *MyType* in interesting ways.

Binary Methods

Difficult to write binary methods with inheritance!

```
class C              {boolean eq(C c){..}}
```

Binary Methods

Difficult to write binary methods with inheritance!

```
class C           {boolean eq(C c){..}}

class D extends C {boolean eq(D d){..}}
class E extends C {boolean eq(E e){..}}
```

Binary Methods

Difficult to write binary methods with inheritance!

```
class C           {boolean eq(C c){..}}

class D extends C {boolean eq(D d){..}}
class E extends C {boolean eq(E e){..}}
```

	Desired	Actual
`new C().eq(new C())`	Ok	Ok
`new C().eq(new D())`	Ok	Ok
`new D().eq(new C())`	Error	Ok
`new E().eq(new D())`	Error	Ok

A Type for `this`

Introduce `ThisClass`: denotes the *class type* of `this`.

With `ThisClass`, can write `eq` as a binary method:

```
class C           {boolean eq(ThisClass tc){..}}
class D extends C {boolean eq(ThisClass tc){..}}
```

New definition: a method is **binary** iff it has a parameter of type `ThisClass`.

A Type for `this`

Introduce `ThisClass`: denotes the *class type* of `this`.

With `ThisClass`, can write `eq` as a binary method:

```
class C            {boolean eq(ThisClass tc){..}}
class D extends C {boolean eq(ThisClass tc){..}}
```

New definition: a method is **binary** iff it has a parameter of type `ThisClass`.

`ThisClass` useful in other situations:

```
public ThisClass clone();
```

Can use a `Factory<ThisClass>` to encode behavior of `This` constructors [Joy].

Example: Linked-list Nodes

```
class Node<T> {
   protected ThisClass next;
   public Node(T t, ThisClass next) {..}
   public ThisClass getNext() {
     return next;
   }
   public void setNext(ThisClass next) {
     this.next = next;
   }
   ..
}
```

Example: Linked-list Nodes

```
class DblNode<T> extends Node<T> {
   protected ThisClass prev;
   public DblNode(T t, ThisClass next, ThisClass prev) {..}
   /* getPrev, setPrev elided */
   public void setNext(ThisClass next) {
     super.setNext(next);
     if (next != null) { next.setPrev(this); }
   }
   ..
}
```

Problems with ThisClass

```
public void breakIt(Node<T> n1, Node<T> n2) {
  n1.setNext(n2);
}
..
Node<T> n;
DblNode<T> dn;
breakIt(dn, n);
```

Problems with ThisClass

```
public void breakIt(Node<T> n1, Node<T> n2) {
  n1.setNext(n2);
}
..
Node<T> n;
DblNode<T> dn;
breakIt(dn, n);

   →* dn.setNext(n) //error!
```

Calls `setNext` on a `DblNode<T>` with an argument of type `Node<T>`, a hole!

Exact Types

- To fix hole, introduce exact types, written `@T`.
- An expression with static type `@T` always refers to an object with run-time type `T` (and not an extension of `T`).
- Restrict binary method invocations to receivers whose type is known exactly.
  (Non-binary methods are typed as in Java).
- Exact types can masquerade as non-exact types:

$$\Delta \vdash @\mathtt{T} <: \mathtt{T}$$

Type Checking Classes

When type checking a class with declaration:

```
class D extends C
```

we assume:

`D extends C` (as always)

```
ThisClass extends D
```

```
this:@ThisClass
```

Type Checking Method Invocations I

Recall `setNext` method from `Node<T>`:

`setNext : @ThisClass` $\longrightarrow$ `void`

- If binary method, receiver must be exact.
- Substitute receiver type for `ThisClass` in signature.

Type Checking Method Invocations I

Recall `setNext` method from `Node<T>`:

```
setNext : @ThisClass → void
```

- If binary method, receiver must be exact.
- Substitute receiver type for `ThisClass` in signature.

```
Node<T> node;
@Node<T> exactNode;
@DblNode<T> exactDblNode;

node.setNext           //error!
exactNode.setNext    : @Node<T> → void
exactDblNode.setNext : @DblNode<T> → void
```

Type Checking Method Invocations II

Recall `getNext` method from `Node<T>`:

```
getNext : () → @ThisClass

Node<T> node;
@Node<T> exactNode;
@DblNode<T> exactDblNode;

node.getNext             : Node<T>
exactNode.getNext        : @Node<T>
exactDblNode.getNext     : @DblNode<T>
```

result loses exactness

Formal Semantics and Implementation

- Proof of type safety for LOOJ core as extension of Featherweight GJ [Igarashi, Pierce, Wadler 99].
  - Models `ThisClass` and exact types (and generics).
  - No interfaces (or assignment).
- Full language implemented as an extension of GJ compiler.
  - Like GJ, translated to standard bytecodes by erasure.
  - But also supports lightweight introspection:
    - Checked type casts.
    - `instanceof` expressions.
    - Arrays still a problem (nowhere to store type).
      - Use wrapper class: `Array<T>`

Exact Types and Interfaces

- What is an exact interface type?
- Can an object with a class type be assigned to an exact interface type?

Exact Types and Interfaces

- What is an exact interface type?
- Can an object with a class type be assigned to an exact interface type?
- Yes, if:
  1. Interface *is exactly* the set of public methods declared in the class.
  2. Class *names* interface as its distinguished exact interface:

```
class C implements @I
```

- If `C` has exact interface `I` then $\Delta \vdash$ `@C<:@I`.

ThisClass and Interfaces

- What does `ThisClass` mean in an interface?

```
interface I {
  boolean eq(ThisClass tc);
}
class C implements @I {
  int x;
  public boolean eq(ThisClass tc) { this.x = tc.x }
}
class D implements @I {
  int y;
  public boolean eq(ThisClass tc) { this.y = tc.y }
}
```

ThisType

Allowing `ThisClass` in interfaces leads to holes:

```
@I i1 = new C();
@I i2 = new D();
i1.eq(i2);
```

ThisType

Allowing `ThisClass` in interfaces leads to holes:

```
@I i1 = new C();
@I i2 = new D();
i1.eq(i2);
```

$\longrightarrow^*$ `new C().eq(new D())`

ThisType

Allowing `ThisClass` in interfaces leads to holes:

```
@I i1 = new C();
@I i2 = new D();
i1.eq(i2);
```

$\longrightarrow^*$ `new C().eq(new D())`

$\longrightarrow^*$ `new D().x //error!`

Our solution:

(1) Forbid uses of `ThisClass` in interfaces.

(2) Introduce `ThisType`: denotes the *interface type* of `this`.

Type Checking Classes and Interfaces

When type checking a class with declaration:

```
class D extends C implements @I
```

we assume:

```
D extends C
D implements @I
ThisClass extends D
this:@ThisClass
ThisType extends I
ThisClass implements @ThisType
```

Example with ThisType

```
interface I {
  boolean eq(ThisType tt);
  int getVal();
}
class C implements @I {
  ..
  public boolean eq(ThisType tt) { this.x == tt.getVal() }
}
class D implements @I {
  ..
  public boolean eq(ThisType tt) { this.y == tt.getVal() }
}
i1.eq(i2);
```

Comparison

LOOM	LOOJ
classes are not types structural type relations *MyType*	class (names) are types named type relations `ThisClass` and `ThisType`
exact by default #-types matching	"slippery" by default @-types `extends`
no type-based operations	checked casts, `instanceof`

Related Work

- Indexicals well studied in linguistics [Kaplan 70s].
- *MyType*, matching: TOOPLE, PolyTOIL, LOOM [Bruce 90s].
- Many proposals for extending Java with generics:
  - Pizza/GJ [Bracha, Odersky, Wadler, Stoutamire 97, 98],
  - NextGen [Allen, Cartwright, Steele, 98],
  - PolyJ [Bank, Liskov, Myers 97],
  - Translation LM [Natali, Viroli 00].
- Early implementation of LOOJ [Burstein].
- Optimized JVM verifier/optimizer for GJ/LOOJ [Gonzalez].

Summary

LOOJ is a conservative extension to Java with:

- `ThisClass`: denotes the class type of `this`.
- `ThisType`: denotes the interface type of `this`.
- Exact types – ensure static and dynamic types agree.

- Formal semantics – Featherweight LOOJ.
- Implementation with lightweight introspection.
- Familiar presentation of many features from LOOM.

Questions?

F-Bounded Polymorphism

With generics can write binary methods, but awkwardly:

```
class C<TC extends C<TC>> { boolean eq(TC tc) { .. } }
class D<TC extends D<TC>> extends C<TC> {
  boolean eq(TC tc) { .. }
}
class ExactC extends C<ExactC> { }
class ExactD extends D<ExactD> { }
```

F-Bounded Polymorphism

With generics can write binary methods, but awkwardly:

```
class C<TC extends C<TC>> { boolean eq(TC tc) { .. } }
class D<TC extends D<TC>> extends C<TC> {
  boolean eq(TC tc) { .. }
}
class ExactC extends C<ExactC> { }
class ExactD extends D<ExactD> { }
```

But tricky, verbose and many types:

```
C<ExactC>     C<ExactD>
ExactC        D<ExactD>
              ExactD
```

In LOOJ: just two classes and `C`, `@C`, `D`, `@D`.

Bounded Polymorphism and ThisType

```
interface I {
  @ThisType getNext();
  void setNext(@ThisType tt);
}
class C,D implements @I {..}
class E<X extends I> {
  private @X x;
  public @X xGetNext() { return x.getNext(); }
}
@C c;
@E<C> e;
c.setNext(new D());
c = e.xGetNext(); //error! (RHS is @D)
```

LOOJ Translation I

```
class C<T> {
  public C() { .. }
    .. obj instanceof C<T> ..

}
new C<String>() ..
```

LOOJ Translation II

```
class C {
  private PolyClass T$$class;
  public C(PolyClass T$$class) {
    this.T$$class = T$$class;
    ..
  }
  public boolean instanceOf$$C(PolyClass T$$class) {
    return this.T$$class.equals(T$$class);
  }
  .. (obj instanceof C)
     && ((C)obj).instanceOf$$C(T$$class) ..
}
new C(new PolyClass(String.class)) ..
```

Featherweight LOOJ: Syntax

Classes	CL	$::=$	$\texttt{class}\ \mathtt{C}\langle\overline{\mathtt{Z}} \triangleleft \overline{\mathtt{N}}\rangle \triangleleft \mathtt{D}\langle\overline{\mathtt{N}}\rangle\ \{\ \overline{\mathtt{T}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\ \}$
Constructors	K	$::=$	$\mathtt{C}(\overline{\mathtt{S}}\ \overline{\mathtt{g}}, \overline{\mathtt{T}}\ \overline{\mathtt{f}})\ \{\ \texttt{super}(\overline{\mathtt{g}});\ \texttt{this}.\overline{\mathtt{f}} = \overline{\mathtt{f}};\ \}$
Methods	M	$::=$	$\langle\overline{\mathtt{Z}} \triangleleft \overline{\mathtt{N}}\rangle\ \mathtt{T}\ \mathtt{m}(\overline{\mathtt{T}}\ \overline{\mathtt{x}})\ \{\ \uparrow \mathtt{e};\ \}$
Expressions	e	$::=$	$\mathtt{x} \mid \mathtt{e.f} \mid \mathtt{x.m}\langle\overline{\mathtt{H}}\rangle(\overline{\mathtt{e}}) \mid \texttt{new}\ \mathtt{C}\langle\overline{\mathtt{H}}\rangle(\overline{\mathtt{e}}) \mid (\mathtt{T})\mathtt{e}$
Types	T	$::=$	$\mathtt{H} \mid @\mathtt{H}$
Hash Types	H	$::=$	$\mathtt{X} \mid \mathtt{C}\langle\overline{\mathtt{H}}\rangle$
Bound Types	N	$::=$	$\mathtt{C}\langle\overline{\mathtt{Z}}\rangle \mid \mathtt{C}\langle\overline{\mathtt{N}}\rangle$

Featherweight LOOJ: Method Type Lookup

$$\frac{CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\langle\overline{\mathtt{Z}} \triangleleft \overline{\mathtt{N}}\rangle \triangleleft \mathtt{D}\langle\overline{\mathtt{U}}\rangle\ \{\ \ldots\ \overline{\mathtt{M}}\ \} \qquad \langle\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{O}}\rangle\mathtt{V}\ \mathtt{m}(\overline{\mathtt{V}}\ \overline{\mathtt{x}})\{\uparrow \mathtt{e};\} \in \overline{\mathtt{M}}}{mtype(\mathtt{m}, \mathtt{C}\langle\overline{\mathtt{T}}\rangle, @\mathtt{R}) = [\overline{\mathtt{T}}/\overline{\mathtt{Z}}][\mathtt{R}/\mathtt{ThisClass}](\langle\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{O}}\rangle\overline{\mathtt{V}} \rightarrow \mathtt{V})}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\langle\overline{\mathtt{Z}} \triangleleft \overline{\mathtt{N}}\rangle \triangleleft \mathtt{D}\langle\overline{\mathtt{U}}\rangle\ \{\ \ldots\ \overline{\mathtt{M}}\ \} \qquad \langle\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{O}}\rangle\mathtt{V}\ \mathtt{m}(\overline{\mathtt{V}}\ \overline{\mathtt{x}})\{\uparrow \mathtt{e};\} \in \overline{\mathtt{M}} \qquad \mathtt{R}\text{ not exact} \qquad \mathtt{ThisClass}\text{ does not appear in }\overline{\mathtt{V}} \qquad pos(\mathtt{V})}{mtype(\mathtt{m}, \mathtt{C}\langle\overline{\mathtt{T}}\rangle, \mathtt{R}) = [\overline{\mathtt{T}}/\overline{\mathtt{Z}}][\mathtt{R}/@\mathtt{ThisClass}, \mathtt{ThisClass}](\langle\overline{\mathtt{Y}} \triangleleft \overline{\mathtt{O}}\rangle\overline{\mathtt{V}} \rightarrow \mathtt{V})}$$

Featherweight LOOJ: Subtyping

$$\Delta \vdash \mathtt{T} <: \mathtt{T}$$

$$\frac{\Delta \vdash \mathtt{S} <: \mathtt{T} \qquad \Delta \vdash \mathtt{T} <: \mathtt{U}}{\Delta \vdash \mathtt{S} <: \mathtt{U}}$$

$$\frac{\mathrm{CT}(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{Z}} \lhd \overline{\mathtt{N}} \rangle \lhd \mathtt{D}\langle \overline{\mathtt{U}} \rangle\ \{\ \ldots\ \}}{\Delta \vdash \mathtt{C}\langle \overline{\mathtt{T}} \rangle <: [\overline{\mathtt{T}}/\overline{\mathtt{Z}}]\mathtt{D}\langle \overline{\mathtt{U}} \rangle}$$

$$\Delta \vdash \mathtt{X} <: \Delta(\mathtt{X})$$

$$\Delta \vdash @\mathtt{T} <: \mathtt{T}$$

Featherweight LOOJ: Expression Typing

$$\frac{\Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \mathit{mtype}(m, \mathrm{bound}_\Delta(\mathtt{T}_0), \mathtt{T}_0) = \langle \overline{\mathtt{Y}} \triangleleft \overline{\mathtt{O}} \rangle \overline{\mathtt{U}} \to \mathtt{U} \\ \Delta \vdash \overline{\mathtt{V}}\ \mathit{ok} \qquad \Delta \vdash \overline{\mathtt{V}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\overline{\mathtt{O}} \qquad \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \qquad \Delta \vdash \overline{\mathtt{S}} <: [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\overline{\mathtt{U}}}{\Delta;\Gamma \vdash \mathtt{e}_0.\mathtt{m}\langle \overline{\mathtt{V}} \rangle (\overline{\mathtt{e}}) : [\overline{\mathtt{V}}/\overline{\mathtt{Y}}]\mathtt{U}}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\langle \overline{\mathtt{Z}} \triangleleft \overline{\mathtt{N}} \rangle \triangleleft \mathtt{D}\langle \overline{\mathtt{U}} \rangle\ \{\ \ldots\ \} \qquad \Delta \vdash \mathtt{C}\langle \overline{\mathtt{T}} \rangle\ \mathit{ok} \\ \Delta;\Gamma \vdash \overline{\mathtt{e}} : \overline{\mathtt{S}} \qquad \mathit{fields}(\mathtt{C}\langle \overline{\mathtt{T}} \rangle, @\mathtt{C}\langle \overline{\mathtt{T}} \rangle) = \overline{\mathtt{R}}\ \overline{\mathtt{f}} \qquad \Delta \vdash \overline{\mathtt{S}} <: \overline{\mathtt{R}}}{\Delta;\Gamma \vdash \mathtt{new}\ \mathtt{C}\langle \overline{\mathtt{T}} \rangle (\overline{\mathtt{e}}) : @\mathtt{C}\langle \overline{\mathtt{T}} \rangle}$$

$$\frac{\Delta \vdash \mathtt{T}\ \mathit{ok} \qquad \Delta;\Gamma \vdash \mathtt{e}_0 : \mathtt{T}_0 \qquad \Delta \vdash \mathtt{T}_0 <: \mathtt{T}}{\Delta;\Gamma \vdash (\mathtt{T})\mathtt{e}_0 : \mathtt{T}}$$

Featherweight LOOJ: Method Typing

$$\frac{\begin{array}{c}\Delta = \overline{\texttt{Z}}\texttt{<:}\overline{\texttt{N}}, \overline{\texttt{Y}}\texttt{<:}\overline{\texttt{O}}, \texttt{ThisClass<:C}\langle\overline{\texttt{Z}}\rangle \\ \Delta \vdash \overline{\texttt{T}}, \texttt{T}, \overline{\texttt{O}}\ ok \qquad \Delta; \overline{\texttt{x}} : \overline{\texttt{T}}, \texttt{this} : \texttt{@ThisClass} \vdash \texttt{e}_0 : \texttt{S} \\ \Delta \vdash \texttt{S<:T} \qquad CT(\texttt{C}) = \texttt{class C}\langle\overline{\texttt{Z}} \triangleleft \overline{\texttt{N}}\rangle \triangleleft \texttt{D}\langle\overline{\texttt{U}}\rangle\ \{\ \ldots\ \} \\ override(\texttt{m}, \texttt{D}\langle\overline{\texttt{U}}\rangle, \langle\overline{\texttt{Y}} \triangleleft \overline{\texttt{O}}\rangle\overline{\texttt{T}} \to \texttt{T})\end{array}}{\langle\overline{\texttt{Y}} \triangleleft \overline{\texttt{O}}\rangle\ \texttt{T}\ \texttt{m}(\overline{\texttt{T}}\ \overline{\texttt{x}})\ \{\ \uparrow \texttt{e}_0;\ \}\ \text{OK in}\ \texttt{C}\langle\overline{\texttt{Z}} \triangleleft \overline{\texttt{N}}\rangle}$$