OGI SCHOOL OF SCIENCE & ENGINEERING

Traits for Java – Early Experiences

Phil Quitslund

23 June 2004

Traits Workshop

OHSU

Traits for Java: The Challenge

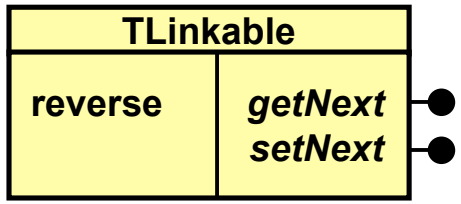
- Smalltalk traits leverage dynamic language features
  - Small,
  - Incomplete,
  - Unbounded units of reuse
- Can we simulate this in Java?

An Example: Linkable Trait

trait named: #TLinkable

reverse

```
| result list temp |
result := temp := nil.
list := self.
[list != nil]
   whileTrue: [
     temp := list getNext.
     list setNext: result.
     result := list.
     list := temp].
^ result.
```



Use case:
A parser where left-recursive productions are built up in Reverse order

Linkable Trait in Java

```
trait TLinkable  {
   ? reverse() {
      ? result = null;
      ? list   = this;
      while (list != null) {
         ? temp = list.getNext();
         list.setNext(result);
         result = list;
         list   = temp ;
      }
      return result;
   }
}
```

Type?

Requires: getNext(): ?

Requires: setNext(?)

Linkable Trait using Interfaces

```
trait TLinkable  {
   ILinkable reverse() {
      ILinkable result = null;
      ILinkable list   = this;
      while (list != null) {
         ILinkable temp = list.getNext();
         list.setNext(result);
         result = list;
         list   = temp ;
      }
      return result;
   }
}
```

```
interface ILinkable {
   setNext(ILinkable l);
   ILinkable getNext();
}
```

```
class VarDecl uses TLinkable{ … }

//in parser:

Decl fields =
  new FieldDecl(vdecls.reverse(), …);
```

Should be of type: VarDecl!

Linkable Trait using ThisType

```
trait TLinkable  {
  ThisType reverse() {
    ThisType result = null;
    ThisType list   = this;
    while (list != null) {
      ThisType temp = list.getNext();
      list.setNext(result);
      result = list;
      list   = temp ;
    }
    return result;
  }
}
```

ThisType keyword gives access to instantiating type

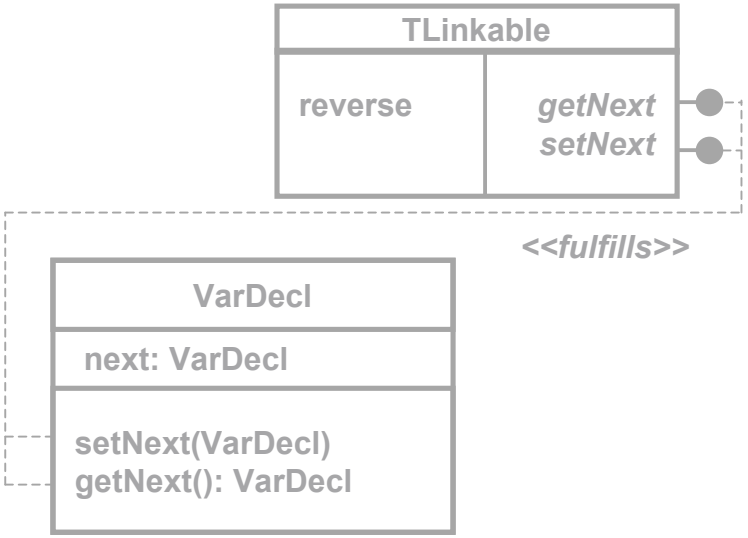

```
class VarDecl uses TLinkable{ … }

//in parser:

Decl fields =
  new FieldDecl(vdecls.reverse(), …);
```

Right type!

Linkable Trait: Requirements?

```
trait TLinkable  {
   ThisType reverse() {
      ThisType result = null;
      ThisType list   = this;
      while (list != null) {
         ThisType temp = list.getNext();
         list.setNext(result);
         result = list;
         list   = temp ;
      }
      return result;
   }
}
```

Q: How to express that TLinkable *requires* methods getNext() and setNext()?



Linkable Trait: Requirements?

```
trait TLinkable  {
  ThisType reverse() {
    ThisType result = null;
    ThisType list   = this;
    while (list != null) {
      ThisType temp = list.getNext();
      list.setNext(result);
      result = list;
      list   = temp ;
    }
    return result;
  }
}
```

Q: How to express that TLinkable *requires* methods getNext() and setNext()?

A_1: Don't! Let the Compiler infer them!

Linkable Trait: Requirements?

```
trait TLinkable  {
  ThisType reverse() {
    ThisType result = null;
    ThisType list   = this;
    while (list != null) {
      ThisType temp = list.getNext();
      list.setNext(result);
      result = list;
      list   = temp ;
    }
    return result;
  }
}
```

Q: How to express that TLinkable *requires* methods getNext() and setNext()?

A_1: Don't! Let the Compiler infer them!

A_2: Declare them.

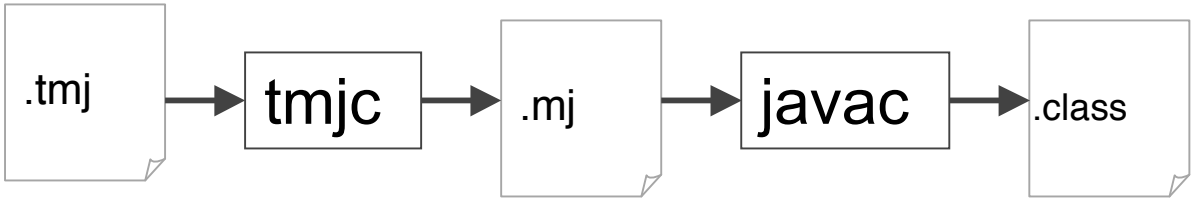
```
trait TLinkable
  requires {ThisType getNext();
            void setNext(ThisType);}
```

Java Traits Design Points

1. Traits should be lightweight
   - Compiler inferred requirements
     - Validation of compositions == constraint satisfaction
   - Optionally, make them explicit
2. Requirements are best expressed structurally
   - Nominal types are not natural
3. Traits are not types
4. Need a ThisType/MyClass mechanism

A Prototype Implementation

- TMJC = TraitsMiniJavaCompiler

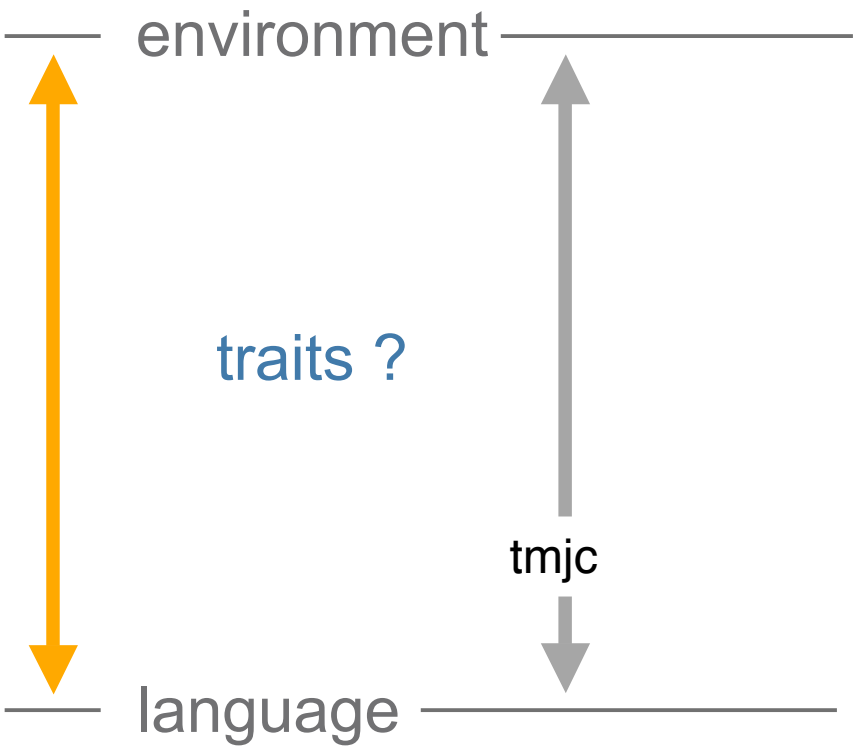


- Works by erasure
- Type checking is done before translation

Javac only sees type-correct pure Java

Pro: control, error-messaging
Con: duplicates logic in javac

Reflections



Less *compiler* hacking
Flattening (for free)
Quicker path to adoption

Base language resistance
Lots of Engineering
Adoption challenges

Open Question:

Is Pure Java Expressive Enough?

- Traits and generics (Java1.5)
  - Binary methods are tricky
    - interface ISortable<T> { boolean lessThan(T t); }
  - Requirements and nominal types
    - class TTrait <ThisType extends SomeIF>
  - Super requirements (expressible?)
  - Keep life easy for the weekend hacker?
    - class SortableList<E extends ISortable<E>>

Thank You!