Morphing: Safely Shaping a Class in the Image of Others

Yannis Smaragdakis
Shan Shan Huang

Department of Computer and Information Science
University of Oregon

IFIP WG 2.11 M5

Motivation: Remove Redundancy

A Synchronization Proxy:

```
class SynchronizedList implements List {
   final List l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Motivation: Remove Redundancy

A Synchronization Proxy:

```
class SynchronizedList implements List {
   final List l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Motivation: Remove Redundancy

A Synchronization Proxy:

```
class SynchronizedList implements List {
   final List l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Motivation: Remove Redundancy

A Synchronization Proxy:

```
class SynchronizedList implements List {
   final List l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Motivation: Remove Redundancy

A Synchronization Proxy:

```
class SynchronizedCollection implements Collection {
   final Collection l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in Collection
}
```

Motivation: Remove Redundancy

A Synchronization Proxy:

```
class SynchronizedSet implements Set {
   final Set l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in Set
}
```

Outline

Synchronization Proxy with MJ:

```
class SynchronizeList implements List {
   final List l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
   final List l;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   public int size () {
      synchronized(mutex) { return l.size();    }
   }
   public boolean remove (int i) {
      synchronized(mutex) { return l.remove(i); }
   }
   ...
   // repeat for all methods in List
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Reflection Set

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Reflective Pattern

Reflection Set

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Reflective Pattern

Reflection Set

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Reflective Pattern

Reflection Set

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Iterator Definition

Reflective Pattern

Reflection Set

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Synchronization Proxy with MJ:

```
class SynchronizeMe<interface T> implements T {
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i) {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i) {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i)   {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i) {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

```
interface List {
   int size ()          {...}
   Object get (int i)   {...}
   void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Parameterizing Proxy:

```
interface List {
   int size ()          {...}
   Object get (int i) {...}
   void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i) {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

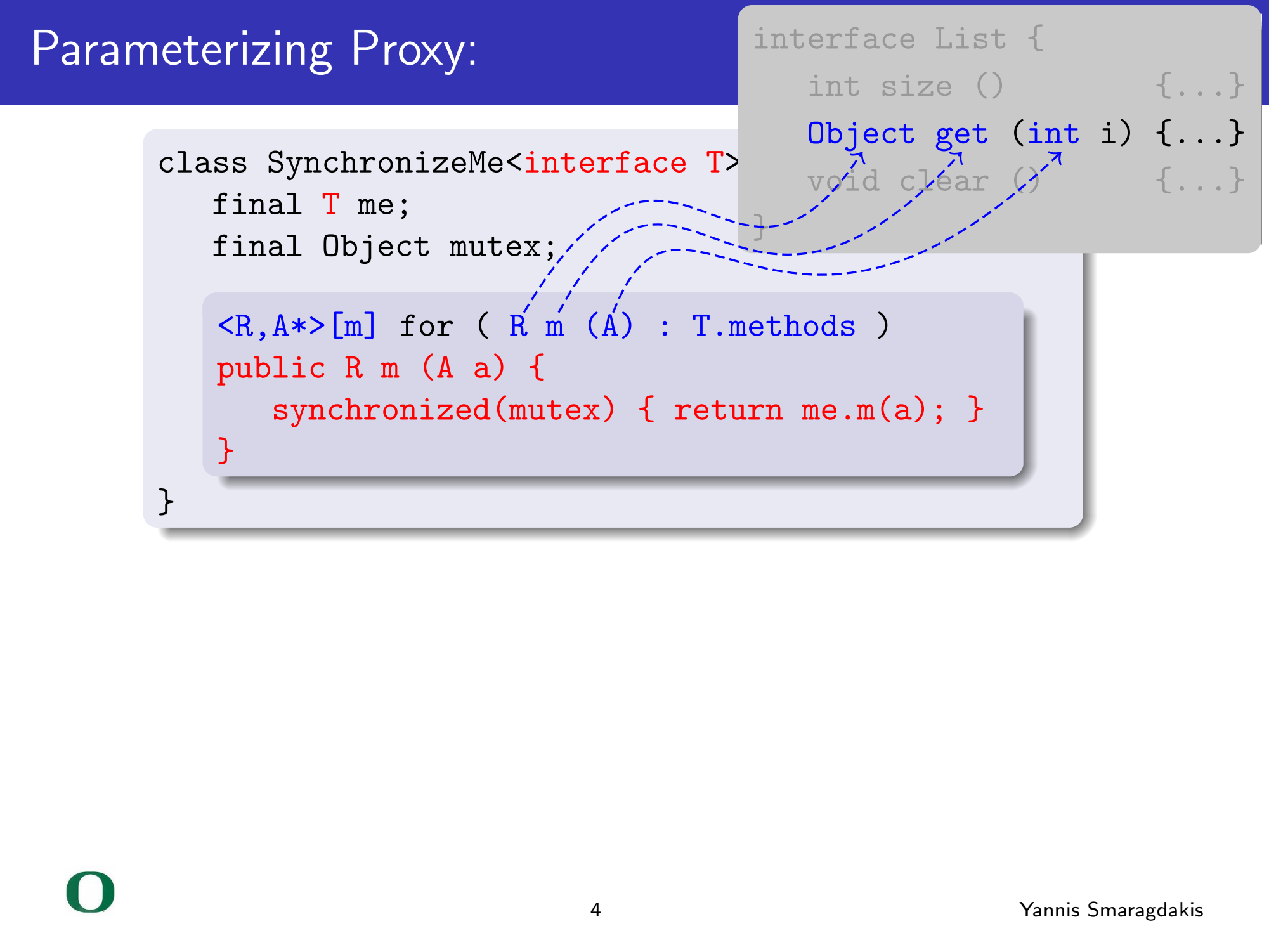
```
interface List {
    int size ()          {...}
    Object get (int i) {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```

Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i)   {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
}
```



Parameterizing Proxy:

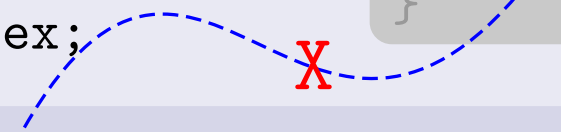
```
interface List {
   int size ()          {...}
   Object get (int i)   {...}
   void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Parameterizing Proxy:

```
interface List {
   int size ()          {...}
   Object get (int i)   {...}
   void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```



Parameterizing Proxy:

```
interface List {
    int size ()          {...}
    Object get (int i)   {...}
    void clear ()        {...}
}
```

```
class SynchronizeMe<interface T>
    final T me;
    final Object mutex;

    <R,A*>[m] for ( R m (A) : T.methods )
    public R m (A a) {
        synchronized(mutex) { return me.m(a); }
    }
```

```
class SynchronizeMeOfList implements List {
    final List me;
    final Object mutex;

    public int size () {
        synchronized(mutex) { return me.size(); }
    }
    public Object get (int a) {
        synchronized(mutex) { return me.get(a); }
    }
}
```

Alternatives?

Alternatives?

- Meta-Object Protocols

Alternatives?

- Meta-Object Protocols
- AOP

Alternatives?

- Meta-Object Protocols
- AOP
- Meta-programming

Alternatives?

- Meta-Object Protocols
- AOP
- Meta-programming

Drawbacks:

- No language synergy.

Alternatives?

- Meta-Object Protocols
- AOP
- Meta-programming

Drawbacks:

- No language synergy.

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Alternatives?

- Meta-Object Protocols
- AOP
- Meta-programming

Drawbacks:

- No language synergy.

```
class SynchronizeMe<interface T> implements T {
   final T me;
   final Object mutex;

   <R,A*>[m] for ( R m (A) : T.methods )
   public R m (A a) {
      synchronized(mutex) { return me.m(a); }
   }
}
```

Alternatives?

- Meta-Object Protocols
- AOP
- Meta-programming

Drawbacks:

- No language synergy.

Alternatives?

- Meta-Object Protocols
- AOP
- Meta-programming

Drawbacks:

- No language synergy.
- No modular type safety!

Outline

1. Morphing with MJ

2. Modularly Type Safe
   - Overview - What Is Modular Type Safety?
   - Validity of References: Range Subsumption
   - Uniqueness of Declarations: Range Disjointness

3. Concluding Remarks

Outline

Outline

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t









}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type Safety?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type

```
class Foo {
   public float bar (int a, int b) {...}
   public int   bar (int a)        {...}
}
```

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type

```
class Foo {
   public float bar (int a, int b) {...}
   public int   bar (int a)        {...}
}
```

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

What Is Modular Type

```
class Foo {
   public float bar (int a, int b) {...}
   public int   bar (int a)        {...}
}
```

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <R,A>[m] for ( public R m (A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.m(arg1);
      return t.m(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

- Unique declaration?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

- Unique declaration?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Challenges in Modular Type Safety

- Unique declaration?
- Valid reference?

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Is This Even Possible????

We want to type-check a class with:

Is This Even Possible????

We want to type-check a class with:

- Unknown number of methods.

Is This Even Possible????

We want to type-check a class with:

- Unknown number of methods.
- Unknown method *names*.

Is This Even Possible????

We want to type-check a class with:

- Unknown number of methods.
- Unknown method *names*.
- Unknown method *type signatures*.

Is This Even Possible????

We want to type-check a class with:

- Unknown number of methods.
- Unknown method *names*.
- Unknown method *type signatures*.
- Unknown supertypes.

Is This Even Possible????

YES!

Key Type Checking Techniques

- Reflectively declared code = Abstract sets

Key Type Checking Techniques

- Reflectively declared code = Abstract sets
- Uniqueness $\Leftrightarrow$ Disjointess
  - Two-way unification of declared signatures.

Key Type Checking Techniques

- Reflectively declared code = Abstract sets
- Uniqueness $\Leftrightarrow$ Disjointess
  - Two-way unification of declared signatures.
- Validity of Reference $\Leftrightarrow$ Subsumption
  - One-way unification from declaration signature to reference signature.

Outline

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Valid Reference

```
class CallWithMax<T> {
   T t;
   ...  // initialize t

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) : S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P
- m ↦ n

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P
- m ↦ n

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P
- m ↦ n

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Reference Between Ranges?

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

- T <: S
- R ↦ P
- m ↦ n
- A ↦ int

```
class ReferToMax<S> {
   CallWithMax<S> max;

   <P>[n] for ( public P n (int, int) :  S.methods )
   public P n ( int arg1, int arg2 ) {
      return max.n(arg1, arg2);
   }
}
```

Outline

Copying Makes You Unique!

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Copying Makes You Unique!

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Copying Makes You Unique!

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Copying Makes You Unique!

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- $R \mapsto P$

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- $R \mapsto P$

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- $R \mapsto P$

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- R $\mapsto$ P
- m $\mapsto$ n

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- R $\mapsto$ P
- m $\mapsto$ n

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- R $\mapsto$ P
- m $\mapsto$ n

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- R $\mapsto$ P
- m $\mapsto$ n
- A $\mapsto$ B

Then there were two...

```
class CallWithMax<T> {
   T t;

   <R,A>[m] for ( public R m (A, A) : T.methods )
   public R m ( A arg1, A arg2 ) {
      return t.m(arg1, arg2);
   }

   <P,B>[n] for ( public P n (B) : T.methods )
   public P n ( B arg1, B arg2 ) {
      if ( arg1.compareTo(arg2) > 0 )
         return t.n(arg1);
      return t.n(arg2);
   }
}
```

- R $\mapsto$ P
- m $\mapsto$ n
- A $\mapsto$ B

Outline

Conclusion

Morphing allows:

- Encapsulate a piece of functionality, adding it uniformly to members of a class.

Conclusion

Morphing allows:

- Encapsulate a piece of functionality, adding it uniformly to members of a class.
- Natural extension of Java Generics.

Conclusion

Morphing allows:

- Encapsulate a piece of functionality, adding it uniformly to members of a class.
- Natural extension of Java Generics.
- Modularly type safe.
  - Type system proven sound.

Questions?