GJ:

Making Java easier to type, and easier to type

Gilad Bracha JavaSoft, Sun Microsystems

Martin Odersky, University of South Australia

David Stoutamire, JavaSoft, Sun Microsystems

Philip Wadler, Bell Labs, Lucent Technologies

Lists in Java and GJ

	in Java	in GJ
integer list	List	List<Integer>
string list	List	List<String>
string list list	List	List<List<String>>

Support for reuse at multiple types

- C++: templates
- Ada: generics
- Standard ML, Haskell: parametric polymorphism
- Java: ???

Collection classes in Java 1.2

- Collection
- Set (HashSet, TreeSet)
- List (ArrayList, LinkedList)
- Map (HashMap, TreeMap)
- Iterator, ListIterator
- Comparator, Comparable

The basics:
Lists in Java and GJ

Lists in Java

```
interface List {
  public void add (Object x);
  public Iterator iterator ();
}
interface Iterator {
  public Object next ();
  public boolean hasNext ();
}
```

Lists in GJ

```
interface List<A> {
  public void add (A x);
  public Iterator<A> iterator ();
}
interface Iterator<A> {
  public A next ();
  public boolean hasNext ();
}
```

Translating lists from GJ to Java

```
interface List {
  public void add (Object x);
  public Iterator iterator ();
}
interface Iterator {
  public Object next ();
  public boolean hasNext ();
}
```

Accessing lists in Java

```
// integer list
List xs = new LinkedList();  xs.add(new Integer(0));
Integer x = (Integer)xs.iterator().next();

// string list list
List ys = new LinkedList();  ys.add(new "zero");
List yss = new LinkedList();  yss.add(ys);
String y = (String)((List)yss.iterator().next()).iterator().next();

// string list treated as integer list
Integer z = (Integer)ys.iterator().next();  // run-time exception
```

Accessing lists in GJ

```
// integer list
List<Integer> xs = new LinkedList<Integer>();  xs.add(new Integer(0));
Integer x = xs.iterator().next();

// string list list
List<String> ys = new LinkedList<String>();  ys.add("zero");
List<List<String>> yss = new LinkedList<List<String>>();  yss.add(ys);
String y = yss.iterator().next().iterator().next();

// string list treated as integer list
Integer z = ys.iterator().next();  // compile-time error
```

Implementing lists in Java

```
class LinkedList implements List {
  protected class Node {
    Object elt;  Node next;
    Node (Object e) { elt=e; next=null; }
  }
  protected Node h, t;
  public LinkedList () { h=new Node(null); t=h; }
  public void add (Object elt) { t.next=new Node(elt); t=t.next; }
  public Iterator iterator () {
    return new Iterator () {
      protected Node p=h.next;
      public boolean hasNext () { return p!=null; }
      public Object next () { Object e=p.elt; p=p.next; return e; }
    };
  }
}
```

Implementing lists in GJ

```
class LinkedList<A> implements List<A> {
  protected class Node {
    A elt;  Node next;
    Node (A e) { elt=e; next=null; }
  }
  protected Node h, t;
  public LinkedList () { h=new Node(null); t=h; }
  public void add (A elt) { t.next=new Node(elt); t=t.next; }
  public Iterator<A> iterator () {
    return new Iterator<A> () {
      protected Node p=h.next;
      public boolean hasNext () { return p!=null; }
      public A next () { A e=p.elt; p=p.next; return e; }
    };
  }
}
```

Generic methods, Bridges, and Bounds

Generic methods: First in Java

```
class Lists {
  public static Object first (List xs) {
    return xs.iterator().next();
  }
}

List ys = new LinkedList();  ys.add("zero");
List yss = new LinkedList<List<String>>();  yss.add(ys);
String y = (String)Lists.first((List)Lists.first(yss));
```

Generic methods: First in GJ

```
class Lists {
  public static <A> A first (List<A> xs) {
    return xs.iterator().next();
  }
}
```

```
List<String> ys = new LinkedList<String>();  ys.add("zero");
List<List<String>> yss = new LinkedList<List<String>>();  yss.add(ys);
String y = Lists.first(Lists.first(yss));
```

Bridges: Comparing bytes in Java

```
interface Comparable {
  public int compareTo (Object that);
}
class Byte implements Comparable {
  private byte value;
  public Byte (byte value) { this.value = value; }
  public byte byteValue () { return value; }
  public int compareTo (Byte that) {  // overloading
    return this.value - that.value;
  }
  public int compareTo (Object that) {  // overriding - bridge
    return this.compareTo((Byte)that);
  }
}
```

Bridges: Comparing bytes in GJ

```
interface Comparable<A> {
  public int compareTo (A that);
}
class Byte implements Comparable<Byte> {
  private byte value;
  public Byte (byte value) { this.value = value; }
  public byte byteValue () { return value; }
  public int compareTo (Byte that) {
    return this.value - that.value;
  }
}
```

Bounds: Maximum in Java

```
class Lists { ...
  public static Comparable max (List xs) {
    Iterator xi = xs.iterator();
    Comparable w = (Comparable)xi.next();
    while (xi.hasNext()) {
      Comparable x = (Comparable)xi.next();
      if (w.compareTo(x) < 0) w = x;
    }
    return w;
  }
}
List xs = new LinkedList(); xs.add(new Byte(0));
Byte x = (Byte)Lists.max(xs);
List ys = new LinkedList(); ys.add(new Boolean(false));
Boolean y = (Boolean)Lists.max(ys);  // run-time exception
```

Bounds: Maximum in GJ

```
class Lists {
  public static <A implements Comparable<A>> A max (List<A> xs) {
    Iterator<A> xi = xs.iterator();
    A w = xi.next();
    while (xi.hasNext()) {
      A x = xi.next();
      if (w.compareTo(x) < 0) w = x;
    }
    return w;
  }
}
List<Byte> xs = new LinkedList<Byte>(); xs.add(new Byte(0));
Byte x = Lists.max(xs);
List<Boolean> ys = new LinkedList<Boolean>(); ys.add(new Boolean(false));
Boolean y = Lists.max(ys);  // compile-time error
```

Retrofitting and Raw types (Evolution *vs.* Revolution)

Review: Old definition with old use

```
// old definition
interface List {
  public void add (Object x);
  public Iterator iterator (); }
interface Iterator {
  public Object next ();
  public boolean hasNext (); }
class LinkedList implements List { ...
  public void add (Object x) { ... }
  public Iterator iterator () { ... } }

// old use
List xs = new LinkedList();  xs.add(new Integer(0));
Integer x = (Integer)xs.iterator().next();
```

Review: New definition with new use

```
// new definition
interface List<A> {
  public void add (A x);
  public Iterator<A> iterator (); }
interface Iterator<A> {
  public A next ();
  public boolean hasNext (); }
class LinkedList<A> implements List<A> { ...
  public void add (A elt) { ... }
  public Iterator<A> iterator () { ... } }

// new use
List<Integer> xs = new LinkedList<Integer>();  xs.add(new Integer(0));
Integer x = xs.iterator().next();
```

Retrofitting types: Old definition with new use

```
// old definition as before
...
// retrofitting
interface List<A> {
  public void add (A x);
  public Iterator<A> iterator (); }
interface Iterator<A> {
  public A next ();
  public boolean hasNext (); }
class LinkedList<A> implements List<A> { ...
  public void add (A elt);
  public Iterator<A> iterator (); }
// new use
List<Integer> xs = new LinkedList<Integer>();  xs.add(new Integer(0));
Integer x = xs.iterator().next();
```

Raw types: New definition with old use

```
// new definition
interface List<A> {
  public void add (A x);
  public Iterator<A> iterator (); }
interface Iterator<A> {
  public A next ();
  public boolean hasNext (); }
class LinkedList<A> implements List<A> { ...
  public void add (A elt) { ... }
  public Iterator<A> iterator () { ... } }

// old use
List xs = new LinkedList();  xs.add(new Integer(0));
Integer x = (Integer)xs.iterator().next();
```

Raw types: Unchecked warnings

```
public String loophole (Integer x) {
  List<String> ys = new LinkedList<String>();
  List xs = ys;
  xs.add(x);  // compile-time unchecked warning
  return ys.iterator().next();
}

// The loophole translated to Java
public String loophole (Integer x) {
  List ys = new LinkedList();
  List xs = ys;
  xs.add(x);
  return (String)ys.iterator().next();  // run-time exception
}
```

Raw types: Equality in Java

```
class LinkedList implements List { ...
  public boolean equals (Object that) {
    if (!that instanceof LinkedList) return false;
    Iterator xi = this.iterator();
    Iterator yi = ((LinkedList)that).iterator();
    while (xi.hasNext() && yi.hasNext()) {
      Object x = xi.next();
      Object y = yi.next();
      if (!(x == null ? y == null : x.equals(y)))
        return false;
    }
    return !xi.hasNext() && !yi.hasNext();
  }
}
```

Raw types: Equality in GJ

```
class LinkedList<A> implements List<A> { ...
  public boolean equals (Object that) {
    if (!that instanceof LinkedList) return false;
    Iterator<A> xi = this.iterator();
    Iterator yi = ((LinkedList)that).iterator();
    while (xi.hasNext() && yi.hasNext()) {
      A x = xi.next();
      Object y = yi.next();
      if (!(x == null ? y == null : x.equals(y)))
        return false;
    }
    return !xi.hasNext() && !yi.hasNext();
  }
}
```

Arrays

Arrays in Java

```
class List { ...
  public int size();
  public Object[] toArray();
}
class LinkedList implements List { ...
  public Object[] toArray() {
    Object[] xa = new Object[this.size()];
    ... // copy list into xa
    return xa;
  }
}

List ys = new LinkedList();  xs.add("zero");
String[] ya = (String[])ys.toArray();  // run-time exception
```

Arrays in GJ — the wrong way

```
class List<A> { ...
  public int size();
  public A[] toArray();
}
class LinkedList<A> implements List<A> { ...
  public A[] toArray() {
    A[] xa = new A[this.size()];  // compile-time unchecked warning
    ... // copy list into xa
    return xa;
  }
}

List<String> ys = new LinkedList<String>();  ys.add("zero");
String[] ya = ys.toArray();  // run-time exception
```

Arrays in GJ — the right way

```
class List<A> { ...
  public int size();
  public A[] toArray(A[] xa);
}
class LinkedList<A> implements List<A> { ...
  public A[] toArray(A[] xa) {
    if (xa.length < this.size())
      xa = gj.lang.Array.newInstance(xa, this.size());  // no warning
    ... // copy list into xa
    return xa;
  }
}
List<String> ys = new LinkedList<String>();  ys.add("zero");
String[] ya = ys.toArray(new String[0]);  // no exception
```

A bridge too far

Iterators in GJ

```
interface Iterator<A> {
  public boolean hasNext ();
  public A next ();
}
class Interval implements Iterator<Integer> {
  private int i;
  private int n;
  public Interval (int l, int u) { i = l; n = u; }
  public boolean hasNext () { return i <= n; }
  public Integer next () { return new Integer(i++); }
}
```

Translating GJ to Java

```
interface Iterator {
  public boolean hasNext ();
  public Object next ();
}
class Interval implements Iterator {
  private int i;
  private int n;
  public Interval (int l, int u) { i = l; n = u; }
  public boolean hasNext () { return i <= n; }
  public Integer next/*1*/ () { return new Integer(i++); }
  // bridge
  public Object next/*2*/ () { return this.next/*1*/(); }
}
```

Conclusion

A Generic Java

- GJ supports generic types

- GJ contains Java as a subset

- GJ compiles to Java, and hence the Java Virtual Machine

- GJ works with Java libraries

Higher-order functions

Lambda calculus

$\lambda\, x.\; x + 1 \;:\; \mathit{int} \;\rightarrow\; \mathit{int}$

SML

```
fn(x) => x+1  :  int -> int
```

GJ

```
interface Function<A,B> {
  public B apply (A x);
}
new Function<Integer,Integer>() {
  public Integer apply (Integer x) {
    return new Integer(x.intValue()+1);
  }
}
```

Related Work

	run-time types	new def old use	old def new use
Pizza Odersky & Wadler	×	×	×
virtual types Thorup	×	×	×
class loader Agesen, Freund, & Mitchell	√	×	×
PolyJ Myers, Bank, & Liskov	√	×	×
NextGen Cartwright & Steele	√	√	×
GJ Bracha, Odersky, Stoutamire, Wadler	×	√	√

GJ vs. C++

	GJ	C++
Requirements on types explicit (bounds)	√	×
Report errors at compile-time	√	×
Avoid code bloat	√	×
Easy to in-line operators	×	√
Allow base types as type parameters	×	√

GJ is available from:

`www.cs.bell-labs.com/~wadler/pizza/gj/`

`www.cis.unisa.edu.au/~pizza/gj/`

`wwwipd.ira.uka.de/~pizza/gj/`

You can give Sun feedback on generic types:

http://developer.javasoft.com/
developer/bugParade/bugs/4173702.html