



# Adding Generics to the Java™ Programming Language

**Gilad Bracha**  
Computational Theologist  
Sun Microsystems

# Goals of This Talk

Familiarize you with the proposed generics extension, as it affects working programmers:

- Basic features and usage

- Migration of pre-existing code

- Current status



# Learning Objectives

You will understand:

What Generic Types and Methods are

Why Generics are useful

How to make the transition to using Generics

How to get your hands on an implementation and give us feedback



# Speaker Qualifications

Specification lead for Generic Extension

Maintainer and Co-author of *Java<sup>TM</sup> Language Specification, 2<sup>nd</sup> edition*

Implemented advanced type systems including generics in the past

Researcher on Object oriented programming languages



# What Are Generics?

Generics abstract over Types

Classes, Interfaces and Methods can be Parameterized by Types

Generics provide increased readability and type safety



# Example

```
interface List<E> {  
    void add(E x);  
    Iterator<E> iterator();  
  
}
```

```
interface Iterator<E> {  
    E next();  
    boolean hasNext();  
  
}
```



# What Generics Are Not

Generics are **not** templates

Unlike C++, generic **declarations** are typechecked

Generics are compiled once and for all

Generic source code not exposed to user

No bloat required



# How to Use Generics

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

Compare with:

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



# List Usage: Without Generics

```
List ys = new LinkedList();
ys.add("zero");
List yss;
yss = new LinkedList();
yss.add(ys);
String y = (String)
    ((List)yss.iterator().next()).iterator().next();
Integer z = (Integer)ys.iterator().next();
// run-time error
```



# List Usage: With Generics

```
List<String> ys = new LinkedList<String>();
ys.add("zero");
List<List<String>> yss;
yss = new LinkedList<List<String>>();
yss.add(ys);
String y =
    yss.iterator().next().iterator().next();
Integer z = ys.iterator().next();
// compile-time error
```



# List Implementation w/o Generics

```
class LinkedList implements List {
    protected class Node {
        Object elt;
        Node next;
        Node(Object elt) {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;
    }
}
```



# List Implementation w/o Generics

```
public Iterator iterator(){
    return new Iterator(){
        protected Node p = h.next;
        public boolean hasNext(){return p != null;}
        public Object next(){
            Object e = p.el;
            p = p.next;
            return e;
        }
    }
}
```



# List Implementation With Generics

```
class LinkedList<E> implements List<E>
    protected class Node {
        E elt;
        Node next;
        Node(E elt){elt = e; next = null;}
    }
    protected Node h, t;
    public LinkedList() {h = new Node(null); t =
h;}
    public void add(E elt){
        t.next = new Node(elt);
        t = t.next;
    }
}
```



# List Implementation With Generics

```
public Iterator<E> iterator() {  
    return new Iterator<E>() {  
        protected Node p = h.next;  
        public boolean hasNext() {return p != null;  
        public E next() {  
            E e = p.el;   
            p = p.next;  
            return e;}}}}}
```



# Generic Methods

```
interface Function<A,B>{ B value(A arg);}
interface Ntuple<T> {
    <S> Ntuple<S> map(Function<T,S> f);
}
```

```
Ntuple<Integer> nti = ....;
nti.map
    (new Function<Integer, Integer>
        {
            Integer value(Integer i) {
                return new Integer(i.intValue()*2);
            }
        }
    );
```



# Bounds: Example Without Generics

```
class ListUtilities {
    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;
    }
}
```



# Bounds: Usage Without Generics

```
List xs = new LinkedList();  
xs.add(new Byte(0));  
Byte x = (Byte) ListUtilities.max(xs);  
List ys = new LinkedList();  
ys.add(new Boolean(false));  
Boolean y = (Boolean) ListUtilities.max(ys);  
    // run-time error
```



# Bounds: Example With Generics

```
class ListUtilities {
    public static <T implements Comparable<T>>
        T max(List<T> xs) {
        Iterator<T> xi = xs.iterator();
        T w = xi.next();
        while (xi.hasNext()) {
            T x = xi.next();
            if (w.compareTo(x) < 0) w = x;
        }
        return w;
    }
}
```



# Bounds: Usage With Generics

```
List<Byte> xs = new LinkedList<Byte>();  
xs.add(new Byte(0));  
Byte x = ListUtilities.max(xs);  
List<Boolean> ys = new LinkedList<Boolean>();  
ys.add(new Boolean(false));  
Boolean y = ListUtilities.max(ys);  
// compile-time error
```



# How Do Generics Affect My Code?

They don't!

Non-generic code can use generic libraries;  
For example, existing code will run unchanged  
with generic Collection library

Painless migration; you can make the API  
generic before converting the implementation



# Raw Types

Allow new, generic definitions to be used by old, non-generic code

```
interface List<E> { ... }
interface Iterator<E>{...}
class LinkedList<E> implements List<E> {...}
// All definitions fully generic, as before
// usage can still be non-generic
List xs = new LinkedList();
xs.add(new Integer(0));
Integer x = (Integer) xs.iterator().next();
```



# 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();  
}
```



# Unchecked Warnings

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



# Retrofitting

Compile usages that rely on generic types with dummy declarations of generic APIs

Example:

```
class LinkedList<E> implements List<E> {  
    public void add(E elt);  
    public Iterator<E> iterator();  
}
```

Supported by retrofitting mode in compiler



# When Can I Start Using Generics?

Public review underway

Plan is to ship in Tiger (2003?)

Early adopters can start now!

Prototype implementation available

Provides drop-in compatibility with  
Java™ Development Kit (JDK™) software



# How Can I Start Using Generics?

Download prototype implementation from  
<http://java.sun.com/people/gbracha/generics-update.html>

Use the compiler as a drop in replacement  
for `javac`.



# Summary of Generics in the Java™ Programming Language

A good way to catch type errors up front

Make your code more readable

None of the C++ template drawbacks

Easy migration path, at your own pace

Totally compatible with current Java

“Early access” available now; should ship with JDK software in the project “Tiger” release



# Credits

Expert group membership:

Gilad Bracha, Sun Microsystems (chair)

Norman Cohen, IBM

Christian Kemper, Inprise

Steve Marx, WebGain

Martin Odersky, EPFL

Sven-Eric Panitz, Software AG

Kresten Thorup, Aarhus University

Philip Wadler, Lucent Technologies



# More Credits

The javac compiler team, past and present

David Stoutamire

Neal Gafter

Iris Garcia

Bill Maddox





# JavaOne<sup>SM</sup>

Sun's 2001 Worldwide Java Developer Conference

# Q&A



**JavaOne**<sup>SM</sup>

Sun's 2001 Worldwide Java Developer Conference™