

# NYC Java Study Groups JUG



January General Meeting – Thursday January 3, 2008

# **NYC Java Study Groups JUG**

**“Generics and Collections”**

**Dario Laverde and Gary Russo**

# “Generics and Collections”

## Sources

- “Generics in the Java Programming Language”  
by Gilad Bracha <http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>
- “Sun Certified Programmer for Java 5 Study Guide”  
by Kathy Sierra and Bert Bates (Osbourne/McGrawHill)
- “Java Generics and Collections”  
by Maurice Naftalin; Philip Wadler (O'Reilly)

# Collections

- Quick review of basics:

overriding:

```
public boolean equals(Object o)
public int hashCode();
```

what is the contract?

is this valid/legal?

```
public int hashCode() { return 42; }
```

# Collections

- **Interfaces:**
  - Collection
  - List
  - Set, SortedSet, NavigableSet
  - Map, SortedMap, NavigableMap
  - Queue
  
  - Comparable
  - Comparator

# Generics

- before:

```
List myIntList = new LinkedList(); // 1
```

```
myIntList.add(new Integer(0)); // 2
```

```
Integer x = (Integer) myIntList.iterator().next(); // 3
```

- after:

```
List<Integer> myIntList = new LinkedList<Integer>(); // 1'
```

```
myIntList.add(new Integer(0)); //2'
```

```
Integer x = myIntList.iterator().next(); // 3'
```

# Generics

- defining generics:

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

- what happens under the covers? does this?:

```
public interface IntegerList {  
    void add(Integer x);  
    Iterator<Integer> iterator();  
}
```

hint: unlike C++'s templates this is implemented with “type erasure”

# Generics and Subtyping

- is this legal?

```
List<String> ls = new ArrayList<String>(); //1
```

```
List<Object> lo = ls; //2
```

- the next following lines:

```
lo.add(new Object()); // 3
```

```
String s = ls.get(0); // 4: attempts to assign an Object to a String!
```

# Wildcards?

- **before:**

```
void printCollection(Collection c) {  
    Iterator i = c.iterator();  
    for (k = 0; k < c.size(); k++) {  
        System.out.println(i.next());  
    }  
}
```

- **after (is this correct?):**

```
void printCollection(Collection<Object> c) {  
    for (Object e : c) {  
        System.out.println(e);  
    }  
}
```

# Wildcards: "?"

- **correct:**

```
void printCollection(Collection<?> c) {  
    for (Object e : c) {  
        System.out.println(e);  
    }  
}
```

- **and:**

```
Collection<?> c = new ArrayList<String>();  
c.add(new Object()); // compile time error
```

# Bounded Wildcards

- **example:**

```
public void drawAll(List<Shape> shapes) {  
    for (Shape s: shapes) {  
        s.draw(this);  
    }  
}
```

This can't unfortunately be called on a `List<Circle>`

- **solution:**

```
public void drawAll(List<? extends Shape> shapes) { ... }
```

Shape is the “upper bound” of the wildcard

# Bounded Wildcards

- **more examples:**

```
public void addRectangle(List<? extends Shape> shapes) {  
    shapes.add(0, new Rectangle()); // compile-time error!  
}
```

```
public class Census {  
    public static void addRegistry(  
        Map<String, ? extends Person> registry) { ...}  
}...  
Map<String, Driver> allDrivers = ...;  
Census.addRegistry(allDrivers);
```

# Generic Methods

- *first attempt:*

```
static void fromArrayToCollection(Object[] a,  
Collection<?> c) {  
    for (Object o : a) {  
        c.add(o); // compile time error  
    }  
}
```

- *refer to Sun's generics tutorial for more*
- *continue now with O'Reilly book:  
“Java Generics”...*

# generics puzzles

- The class Foo requires that the parameterized type have multiple bounds.  
Which is the correct syntax for declaring the generic class, Foo?
- a) `class Foo<T extends TypeA<? super T> & TypeB>`
- b) `class Foo<T super ? & T extends TypeB>`
- c) `class Foo<T extends TypeA<? super T> && TypeB>`
- d) `class Foo<? super T && T extends TypeB>`
- e) none of the above, you cannot declare a parameterized type with multiple bounds.

# generics puzzles

- Answer: a

Yes, you can have multiple bounds, as long as the second one is an interface (because you cannot have multiple inheritance of classes). Multiple bounds are specified with a single ampersand. b) is incorrect because you cannot have a wild card '?' as the lower or upper bound for a specified type T (the reverse is allowed; the wild card type can have a lower or upper bound of T).

- Given the following code select which statements are true:

```
1 import java.util.*;
2 class MyGenericType<T extends MyGenericType<T>> {
3 }
4 public class TestMyGenericTypes {
5     class MySubGenericType extends MyGenericType {
6     }
7     class MySubGenericType2 extends MyGenericType<MySubGenericType> {
8     }
9     class MySubGenericType3 extends MyGenericType<MySubGenericType3> {
10 }
11 public static void main(String args) {
12     MyGenericType<Integer> foo;
13     MySubGenericType2<MyGenericType2> foo2;
14     MySubGenericType3<MyGenericType> foo3;
15 }
16 }
```

- a) program will not compile, error in line 2
- b) program will not compile, error in line 5
- c) program will not compile, error in line 7
- d) program will not compile, error in line 9
- e) program will not compile, error in line 12
- f) program will not compile, error in line 13
- g) program will not compile, error in line 14
- h) program compiles, but run time error due to infinite recursion

- Answers: c e f g

Here is the compiler output:

```
javac TestMyGenericTypes.java
```

```
TestMyGenericTypes.java:7: type parameter  
TestMyGenericTypes.MySubGenericType is not within its bound
```

- ```
class MySubGenericType2 extends MyGenericType<MySubGenericType> {  
^
```

```
TestMyGenericTypes.java:12: type parameter java.lang.Integer is not within  
its bound
```

```
MyGenericType<Integer> foo;  
^
```

```
TestMyGenericTypes.java:13: type  
TestMyGenericTypes.MySubGenericType2 does not take parameters  
MySubGenericType2<MySubGenericType2> foo2;
```

```
^
```

```
TestMyGenericTypes.java:14: type  
TestMyGenericTypes.MySubGenericType3 does not take parameters  
MySubGenericType3<MySubGenericType> foo3;
```

- The program does not successfully compile, although line 2 does:

```
class MyGenericType<T extends MyGenericType<T>>
```

Which appears somewhat recursive but simply restricts the type to be of a parameterized bound of itself. This is the actual signature used by Enum. For more info refer to the IBM Generics Tutorial.

Line 5 compiles because you can extend a generic type in a "raw type" way for backwards compatibility purposes but this is not recommended.

Line 7 fails because of the strict bounds placed in line 2.

Line 9 does not fail, because it does pass the correct type parameter.

Line 12 fails because of the incorrect type parameter.

Lines 13 and 14 fail because the subclasses were not defined to take type parameters.

# spec bug and trick

- Documentation bug in JLS Section 4.11

```
void <S> loop(S s){ this.<S>loop(s);}
```

should be:

```
<S> void loop(S s){ this.<S>loop(s);}
```

- You can't do the following:

```
g.doSomething(Collections.emptyList());
```

but you can do this:

```
g.doSomething(Collections.<String>emptyList());
```

source: <http://stuffthathappens.com/blog/2007/11/07>

# Java 7 Proposals

- Generics related proposals:
  - improved type inference: argument positions

(from previous example):

```
g.doSomething(Collections.emptyList()); //now compiles
```

- improved type inference: constructors

```
Map<String, List<String>> anagrams =  
    new HashMap<String, List<String>>();
```

becomes:

```
Map<String, List<String>> anagrams = new HashMap<>();
```