The Collections Connection
Ninth Edition (Really!)

Josh Bloch
Google

Martin Buchholz
Sun Microsystems

BOF-0498
Goal of Talk
Find true happiness

Achieve world domination with the Java™ Collections Framework
Outline

I. What’s New in Tiger? (quick review)
   An awful lot! (JSR-14, JSR-201, JSR-166, etc.)

II. What Coming in Mustang?
   More cool stuff
   Some dull stuff

III. Q & A
   Pearls of wisdom from the assembled multitudes
I. What's (Relatively) New in Tiger?

- Three language features
  - Generics, For-each loop, Autoboxing
- Three core collection interfaces
  - Queue, BlockingQueue, ConcurrentHashMap
- One skeletal implementation
  - AbstractQueue
- Eleven (!) concrete implementations
  - 2 Queue, 5 BlockingQueue, 2 Map, 2 Set
- A handful of generic algorithms and such
Language Feature – Generics

- Provides compile-time type safety for collections and eliminates drudgery of casting
  - Tell compiler the element type of collection
  - Compiler inserts casts for you
  - Casts won't fail at runtime
  - “Stronger typing with less typing”

```java
// Removes 4-letter words from c
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
        if (i.next().length() == 4)
            i.remove();
}
```
Language Feature – For-Each Loop

- Eliminates drudgery, error-proneness of iterators
  - Tell compiler what collection you want to traverse
  - Compiler takes care of iterator or index for you
  - You won't make cut-and-paste errors

```java
void cancelAll(Collection<TimerTask> c) {
    for (TimerTask task : c)
        task.cancel();
}
```
Language Feature – Autoboxing

- Eliminates the drudgery of manual conversion between primitive types (such as `int`) and wrapper types (such as `Integer`)

```java
class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m =
            new TreeMap<String, Integer>();

        for (String word : args) {
            Integer freq = m.get(word);
            m.put(word, (freq == null ? 1 : freq + 1));
        }
        System.out.println(m);
    }
}
```
New Interfaces

- **Queue** - Collection that holds elements for processing
- **BlockingQueue** - Queue that allows client to wait for an element to appear ("work queue")
- **ConcurrentMap** - Map that facilitates concurrent use
Queue Interface extends Collection

Throws exception    Returns special value

<table>
<thead>
<tr>
<th>Insert</th>
<th>add(e)</th>
<th>offer(e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td>remove()</td>
<td>poll()</td>
</tr>
<tr>
<td>Examine</td>
<td>element()</td>
<td>peek()</td>
</tr>
</tbody>
</table>
Sample Use of Queue Interface

```java
static <E> List<E> heapSort(
    Collection<E extends Comparable<? Super E>> c) {
    Queue<E> queue = new PriorityQueue<E>(c);
    List<E> result = new ArrayList<E>();
    while (!queue.isEmpty())
        result.add(queue.remove());
    return result;
}
```

In practice you should simply call `Collections.sort`
## BlockingQueue extends Queue

<table>
<thead>
<tr>
<th>Exception</th>
<th>Special-value</th>
<th>Blocks</th>
<th>Times out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>add(e)</td>
<td>offer(e)</td>
<td>put(e)</td>
</tr>
<tr>
<td>Remove</td>
<td>remove()</td>
<td>poll()</td>
<td>take()</td>
</tr>
<tr>
<td>Examine</td>
<td>element()</td>
<td>peek()</td>
<td>N/A</td>
</tr>
</tbody>
</table>
ConcurrentMap Interface

Adds atomic “mini-transactions” to Map

```java
// Insert entry for key if none present
V putIfAbsent(K key, V value);

// Remove entry for key if mapped to value
boolean remove(Object key, Object value);

// Replace entry for key if present
V replace(K key, V newValue);

// Replace entry for key if mapped to oldVal
boolean replace(K key, V oldVal, V newVal);
```
Queue Implementations

- **LinkedQueue** - basic concurrent FIFO queue
- **PriorityQueue** - heap-based priority queue
- **LinkedList** - retrofitted to implement Queue
- **AbstractQueue** - skeletal implementation
BlockingQueue Implementations

- LinkedBlockingQueue - basic FIFO impl
- ArrayBlockingQueue - fixed size impl
- PriorityBlockingQueue - what you'd expect
- DelayQueue - “scheduling” queue
- SynchronousQueue - rendezvous mechanism
ConcurrentMap Implementation

- ConcurrentHashMap
  - Reads never block!
  - Choose write concurrency level at create time
  - Drop-in replacement for Hashtable*
  - Iterators weakly consistent rather than fail-fast
  - No way to lock entire table
  - Prohibits null keys and values

* in programs that rely on thread safety but not on synchronization details
Map and Set Implementations

- **EnumSet** - element type must be enum
  - Uses bit-vector representation internally
  - `long` or `long[]` depending on cardinality of enum
  - Bloody fast

- **EnumMap** - key type must be enum
  - Uses array representation internally
  - Bloody fast
Type-safety of Generics Has Its Limits

- Generics implemented by *erasure*
- Automatically generated casts may fail when interoperating with legacy (or malicious) clients

```java
public class New {
    public static void main(String[] args) {
        List<String> listOfString = new ArrayList<String>();
        Old.putInteger(ls);
        System.out.println(listOfString.get(0).toUpperCase());
    }
}

public class Old {
    public static void putInteger(List list) {
        list.add(new Integer(42));
    }
}
```
New Convenience Implementations – Checked Collection Wrappers

- Guarantee runtime type-safety

```java
Set<String> s = Collections.checkedSet(
    new HashSet<String>(), String.class);
```

- Wrappers provided for all collection interfaces
- Very useful for debugging
Generic Algorithms

- `frequency(Collection<?> c, Object o)`
  - Counts the number of times `c` occurs in `o`

- `disjoint(Collection<?> c1, Collection<?> c2)`
  - Determines whether two collections are disjoint

- `addAll(Collection<? super T> c, T... a)`
  - Adds all of the elements in the array `a` to `c`
    
    ```java
    Collections.addAll(stooges, "Larry", "Moe", "Curly");
    ```

- `reverseOrder(Comparator<T> cmp)`
  - Returns comparator representing reverse ordering of `cmp`
Utility Methods – java.util.Arrays

- Content-based `equals` present since 1.2
- Added `hashCode`, `toString` to go along
  - No more `Arrays.asList` to print arrays!
    ```java
    System.out.println(Arrays.toString(myArray));
    ```
  - Useful for writing `hashCode` and `toString` methods on classes containing arrays
- For multidimensional arrays: `deepEquals`, `deepHashCode`, `deepToString`
  ```java
  System.out.println(
    Arrays.deepToString(myMatrix));
  ```
Miscellany – Bit Twiddling

- Common bit-manipulation operations for primitives `Integer`, `Long`, `Short`, `Byte`, `Char`
- `highestOneBit`, `lowestOneBit`
- `numberOfLeadingZeros`, `numberOfTrailingZeros`
- `bitCount`
- `rotateLeft`, `rotateRight`
- `reverse`, `reverseBytes`
- `signum`
If You Love Bit Twiddling, Buy This Book!

Hacker's Delight

Henry S. Warren, Jr.

1111^2 = 11100001

pop(x) = \sum_{i=0}^{\lfloor \log_2 x \rfloor} (-1)^i

George Boole
1815 - 1864

n = -2^{h_1} + 2^{h_2} + 2^{h_3} + \ldots + 2^{h_{b}}

\frac{1}{2} = 0.01010101...

x \oplus y = (x | y) - (x \& y)

x + y = (x | y) + (x \& y)

x - y = x + \bar{y} + 1

[x] = \lceil -x \rceil

\bar{x} = x + 1

Nam factors of 2 in x = \log_2(x \& (-x)), x \neq 0

2^{3} + 1 = 641 \cdot 6700417

2^{4} + 1 = 274177 \cdot 67280421310721

\lfloor 1111111 \rfloor = 1111

[a] + [b] \leq [a + b] \leq [a] + [b] + 1

p_n = 1 + \frac{2\pi^2}{6n} \left[ \sum_{j=1}^{\infty} \frac{\sin \pi(n-1)j + 1}{\pi^2 n} \right]^{1/n}
Collections in Java SE 6 ("Mustang")

Bidirectional collections

- Deques
- Navigable collections

Fewer features, but…

- More bug-fixing
- More accurate API docs
- More community involvement
Focus on Bug-fixing

Our favorite bug fix

5045582: binarySearch fails when size() greater than 1**30

- int mid = (low + high) >> 1;
+ int mid = (low + high) >>> 1;
## Interface Deque extends Queue

<table>
<thead>
<tr>
<th></th>
<th>First Element (Head)</th>
<th>Last Element (Tail)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>exception</td>
<td>special value</td>
</tr>
<tr>
<td>Insert</td>
<td><code>addFirst(e)</code></td>
<td><code>offerFirst(e)</code></td>
</tr>
<tr>
<td>Remove</td>
<td><code>removeFirst()</code></td>
<td><code>pollFirst()</code></td>
</tr>
<tr>
<td>Examine</td>
<td><code>getFirst()</code></td>
<td><code>peekFirst()</code></td>
</tr>
</tbody>
</table>
# Deque - Queue Equivalents

<table>
<thead>
<tr>
<th>Queue Method</th>
<th>Equivalent Deque Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>offer(e)</td>
<td>offerLast(e)</td>
</tr>
<tr>
<td>add(e)</td>
<td>addLast(e)</td>
</tr>
<tr>
<td>poll()</td>
<td>pollFirst()</td>
</tr>
<tr>
<td>remove()</td>
<td>removeFirst()</td>
</tr>
<tr>
<td>peek()</td>
<td>peekFirst()</td>
</tr>
<tr>
<td>element()</td>
<td>getFirst()</td>
</tr>
</tbody>
</table>
Using a Deque as a Stack

<table>
<thead>
<tr>
<th>Stack Method</th>
<th>Equivalent Deque Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(e)</td>
<td>addFirst(e)</td>
</tr>
<tr>
<td>pop()</td>
<td>removeFirst()</td>
</tr>
<tr>
<td>peek()</td>
<td>peekFirst()</td>
</tr>
</tbody>
</table>
Interface BlockingDeque extends BlockingQueue

First Element (Head)

<table>
<thead>
<tr>
<th>Block</th>
<th>Time out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>putFirst(e)</td>
</tr>
<tr>
<td></td>
<td>offerFirst(e, time, unit)</td>
</tr>
<tr>
<td>Remove</td>
<td>takeFirst()</td>
</tr>
<tr>
<td></td>
<td>pollFirst(time, unit)</td>
</tr>
</tbody>
</table>

Last Element (Tail)

<table>
<thead>
<tr>
<th>Block</th>
<th>Time out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>putLast(e)</td>
</tr>
<tr>
<td></td>
<td>offerLast(e, time, unit)</td>
</tr>
<tr>
<td>Remove</td>
<td>takeLast()</td>
</tr>
<tr>
<td></td>
<td>pollLast(time, unit)</td>
</tr>
</tbody>
</table>
Deque and BlockingDeque Implementations

- **ArrayDeque** - basic Deque implementation
  - Very fast (implemented as circular buffer)
  - Stack and queue of choice
  - (Perhaps in dolphin) **List** of choice?

- **LinkedBlockingDeque** - highly concurrent BlockingDeque implementation

- **LinkedList** - retrofitted to implement Deque
Deque Example

- McDonalds Drive-Thru
  - (and no, I don’t eat there)
- You arrive at the end of the line
- You leave from the beginning of the line...
- …Unless the line is too long, in which case you can leave from the end (but not the middle!)
- (Many more serious uses as well)
Navigable Collections

- Add methods that should have been there in the first place
- `NavigableSet` extends `SortedSet`
- `NavigableMap` extends `SortedMap`
- Use `NavigableXxx` in place of `SortedXxx` in new work

- `ConcurrentNavigableMap` extends `ConcurrentMap` and `NavigableMap`
  - Takes advantage of covariant returns for `Map` views
NavigableSet Interface - Navigation

// Returns least element >= to given element, or null
E ceiling(E e);

// Returns least element > given element, or null
E higher(E e);

// Returns greatest element <= given element, or null
E floor(E e);

// Returns greatest element < given element, or null
E lower(E e);

// Gets and removes the first (lowest) element, or null
E pollFirst();

// Gets and removes the last (highest) element, or null
E pollLast();
NavigableSet Interface - Views

// Returns descending view of set
NavigableSet<E> descendingSet();

// Returns view: fromElement -> toElement
NavigableSet<E> subSet(E fromElement, boolean fromInc,
                      E toElement, boolean toInc);

// Returns view: <beginning> -> toElement
NavigableSet<E> headSet(E toElement, boolean inclusive);

// Returns view: fromElement -> <end>
NavigableSet<E> tailSet(E fromElement, boolean inclusive);

// Returns iterator in descending order
Iterator<E> descendingIterator();
A Small Example Use of NavigableSet

- Words from "cat" to "dog" (inclusive)
  - Before: `sortedSet.subSet("cat", "dog" + '\0')`
  - After: `navigableSet.subSet("cat", true, "dog", true)`

- BigDecimals from 1 to 10 (inclusive)
  - Before: Not Possible!!
  - After: `navigableSet.subSet(1, true, 10, true)`

- NavigableSet fixes many other deficiencies in SortedSet as well
NavigableMap Interface
Obvious Analogue of NavigableSet

- ceilingEntry(K key), ceilingKey(K key)
- higherEntry(K key), higherKey(K key)
- floorEntry(K key), floorKey(K key)
- lowerEntry(K key), lowerKey(K key)
- firstEntry(), pollFirstEntry()
- lastEntry(), pollLastEntry()
- descendingMap(), descendingKeySet()
- subMap(K, boolean, K, boolean),
  headMap(K toKey, boolean inclusive),
  tailMap(K fromKey, boolean inclusive)
Navigable Collection Implementations

- TreeSet retrofitted for NavigableSet
- TreeMap retrofitted for NavigableMap
- ConcurrentSkipListSet implements NavigableSet
- ConcurrentSkipListMap implements ConcurrentNavigableMap
Arrays.copyOf

Before:

```java
int[] newArray = new int[newLength];
System.arraycopy(oldArray, 0, newArray, 0, oldArray.length);
```

After:

```java
int[] newArray = Arrays.copyOf(a, newLength);
```
Collections.newSetFromMap

The JDK does not provide an IdentityHashSet class, but...

```java
Set<Object> identityHashSet = Collections.newSetFromMap(
    new IdentityHashMap<Object, Boolean>());
```
AbstractMap.SimpleEntry (finally)

- Writing your own `Map` used to be a pain
  - You had to roll your own `Map.Entry` from scratch
  - Not any more!
- `AbstractMap.SimpleEntry` is a fully functional, concrete `Map.Entry` implementation
- Not earthshaking, but a real convenience
Josh’s Java 7 (“Dolphin”) Wish List

- Builder<T>
- ReferenceMap/Cache
- Multiset
- Multimap
- BiMap
- Forwarding{Collection, Set, List, Map}
- AbstractIterator
- Fast String Collections / Algorithms
Martin’s Java 7 (“Dolphin”) musings

- **ScalableArrayDequeList?**
  - ArrayList `get(int)` very fast, but ...
  - `remove(int), add(int)`, wasted space O(n)
- **ValueWeakIdentityHashMap?**
- **CopyOnWriteArrayHashSet?**
- **ScalableIntSet?**
  - **BitSet** too specialized, name is misleading
Useful URLs

- Collections framework enhancements in Tiger
  - http://java.sun.com/j2se/5.0/docs/guide/collections/changes5.html

- Collections API, Tutorial, etc.
  - http://java.sun.com/j2se/5.0/docs/guide/collections

- Mustang Collections
  - http://gee.cs.oswego.edu/dl/concurrency-interest
  - http://download.java.net/jdk6/docs/api
Community Shout-Out

- Props to these homies
  - Doug Lea
  - David Holmes (Now at Sun)
  - Jason Mehrens
  - Tom Hawtin
  - Holger Hofstätte
  - Anyone we forgot
A book from some friends

- Collections and Concurrency are inseparable
- The practical Java concurrency book
- From the folks who brought you `java.util.concurrent`
Not just for fun

- 95 Puzzles
- 52 Illusions
- Collections
Coming soon (?)

- Collections and Generics are inseparable
- The practical Java Generics book
- From some of the folks who designed Java Generics
Obligatory Graphic
Q&A

Joshua Bloch
Martin Buchholz
Our Imaginary Friend Herbert
The Collections Connection
Ninth Edition (Really!)

Josh Bloch
Google

Martin Buchholz
Sun Microsystems

BOF-0498