Collections API

Alex Miller, Terracotta

http://tech.puredanger.com

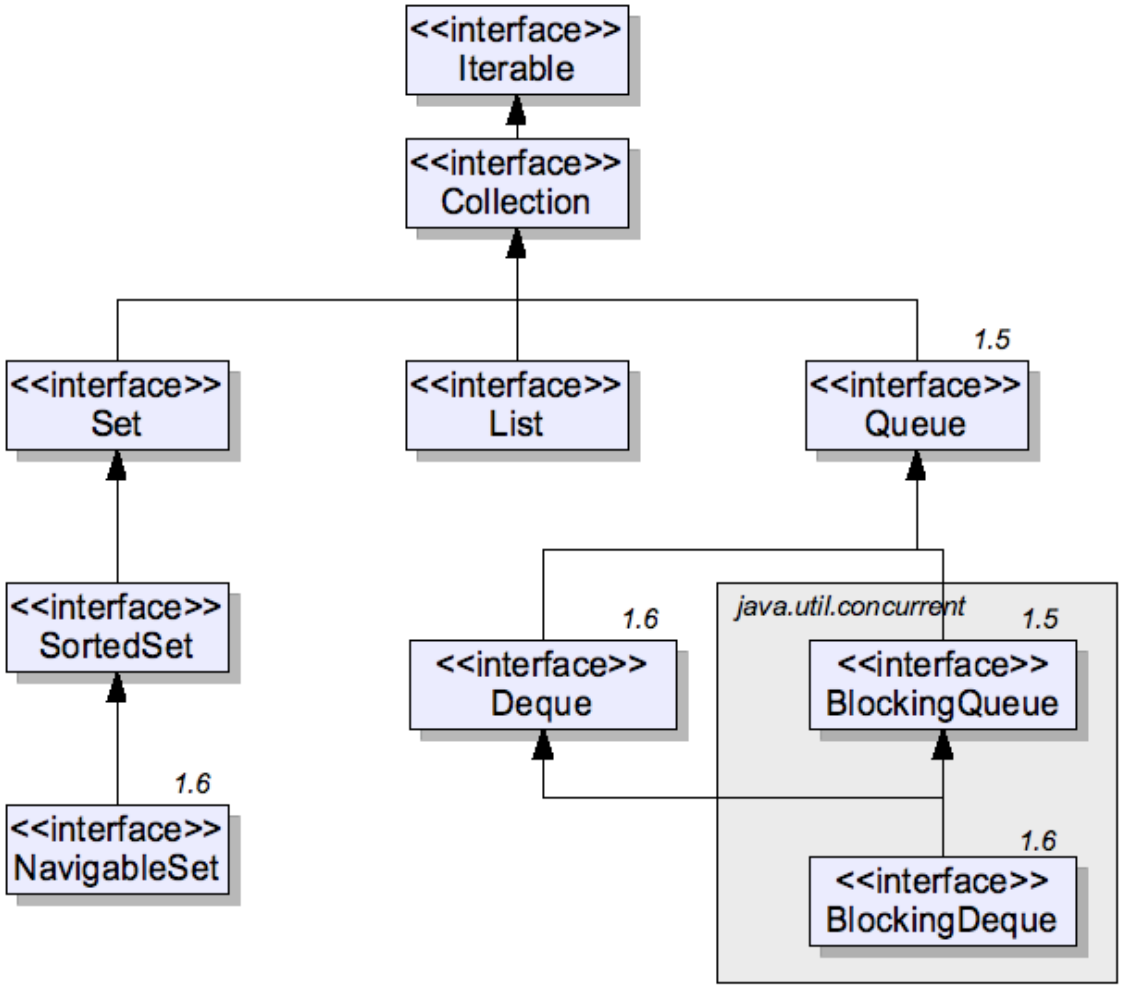
Topics

- Data structures
- Axes of comparison
- Collection interfaces
- Collection implementations
- Algorithms
- Concurrency

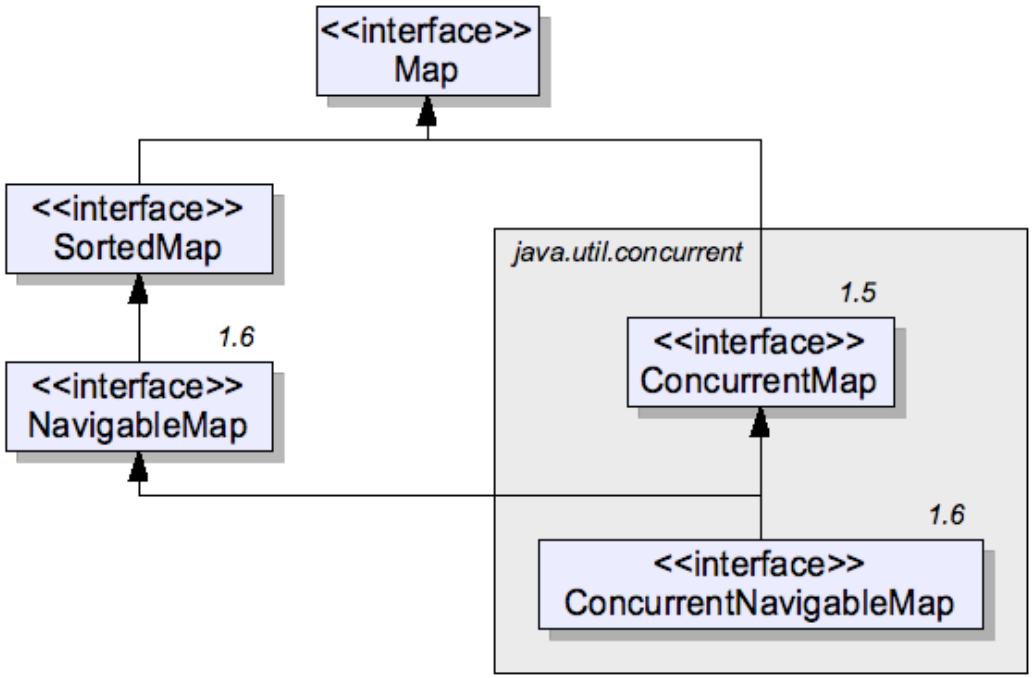
History

- JDK 1.0: Vector, Dictionary, Hashtable, Stack, Enumeration
- JDK 1.2: Collection, Iterator, List, Set, Map, ArrayList, HashSet, TreeSet, HashMap, WeakHashMap
- JDK 1.4: RandomAccess, IdentityHashMap, LinkedHashMap, LinkedHashSet
- JDK 1.5: Queue, java.util.concurrent, ...
- JDK 1.6: Deque, ConcurrentSkipListSet/Map, ...
- JDK 1.7: TransferQueue, LinkedTransferQueue

Collection Interfaces



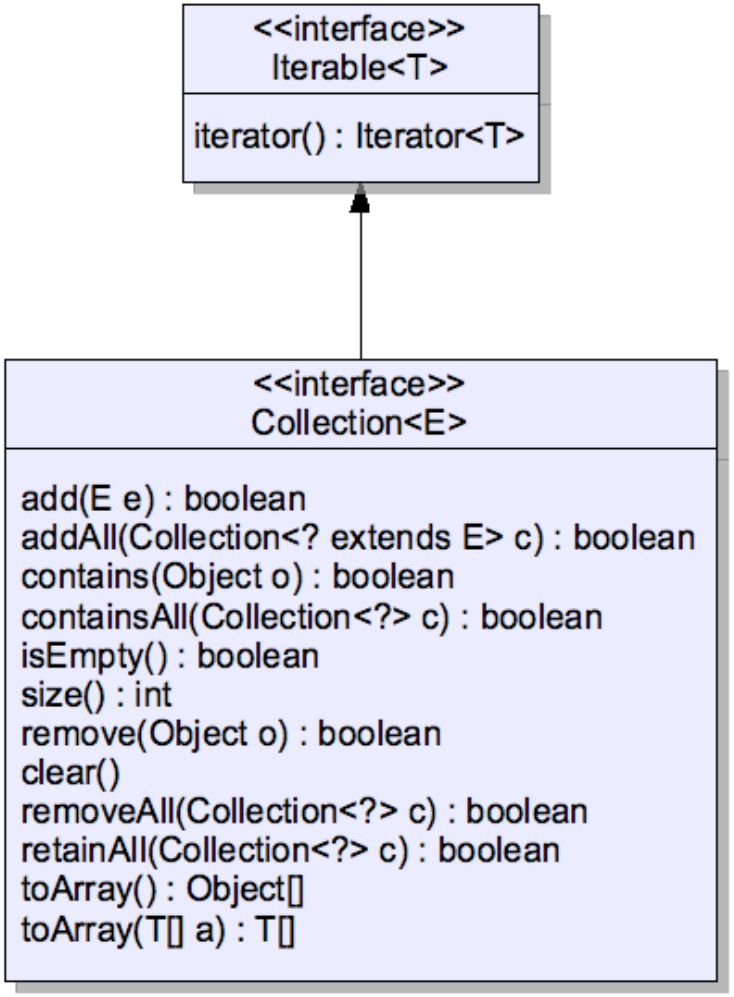
Map Interfaces



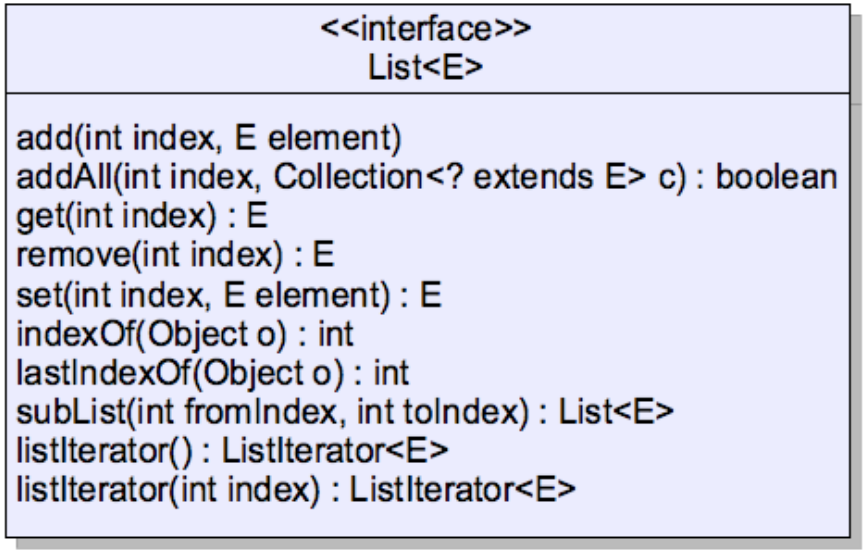
Comparing Implementations

- Operation performance
- Concurrency
- Iterator style
- Data types
- Sorting and ordering
- Null support

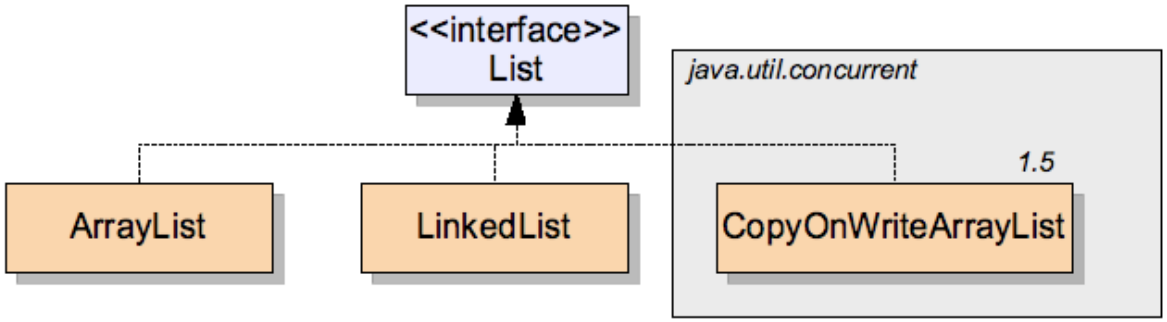
Collection Interface



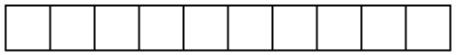
List Interface



List Implementations

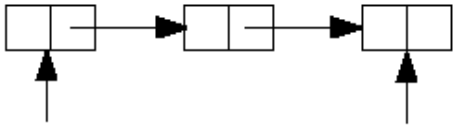


Data Structure: Array



- Indexed access
- Uses offset from memory address for fast memory access
- In Java - fixed size memory block with VM-level support

Data structure: Linked List

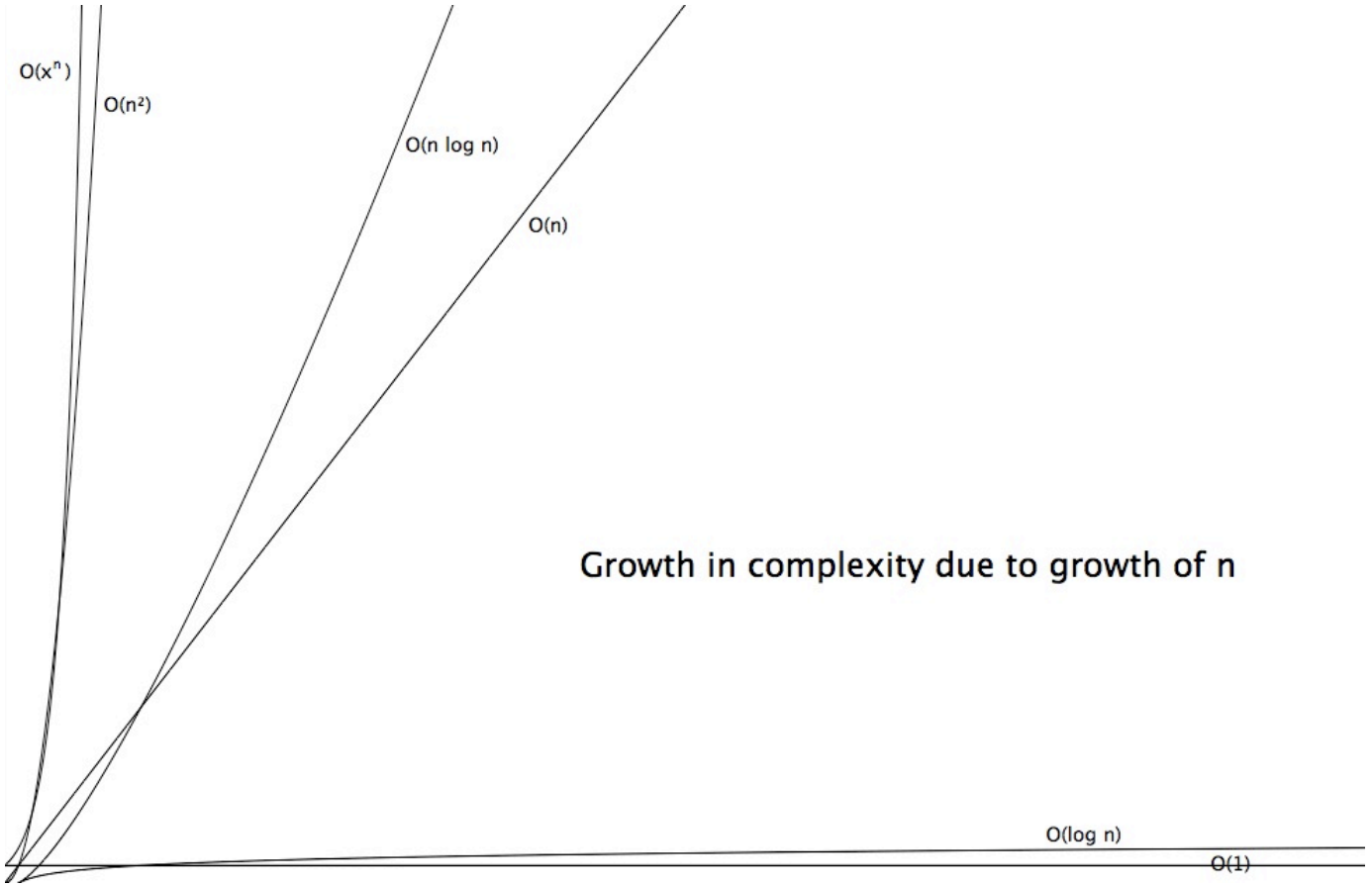


- Dynamic structure made of pointers
- Easy to insert and delete new elements
- No indexed access

List Comparison

	Data Structure	Sorting	Iterator	Nulls?
Array List	Array	No	Fail-fast	Yes
Linked List	Linked list	No	Fail-fast	Yes
CopyOnWrite ArrayList	Array	No	Snapshot	Yes

Computational Complexity



List Performance Comparison

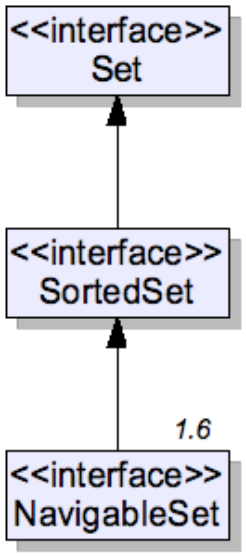
	add	remove	get	contains
ArrayList	O(1)	O(n)	O(1)	O(n)
LinkedList	O(1)	O(1)	O(n)	O(n)
CopyOnWrite ArrayList	O(n)	O(n)	O(1)	O(n)

Iterators

- Fail-fast - work on live data, but become invalid when live data is modified
- Weakly consistent - work on live data, safe, reflect updates and deletes, but not inserts
- Snapshot - work on a snapshot of the live data, fast, safe, but possibly stale

Demo

Set Interfaces



SortedSet

first() : E
last() : E

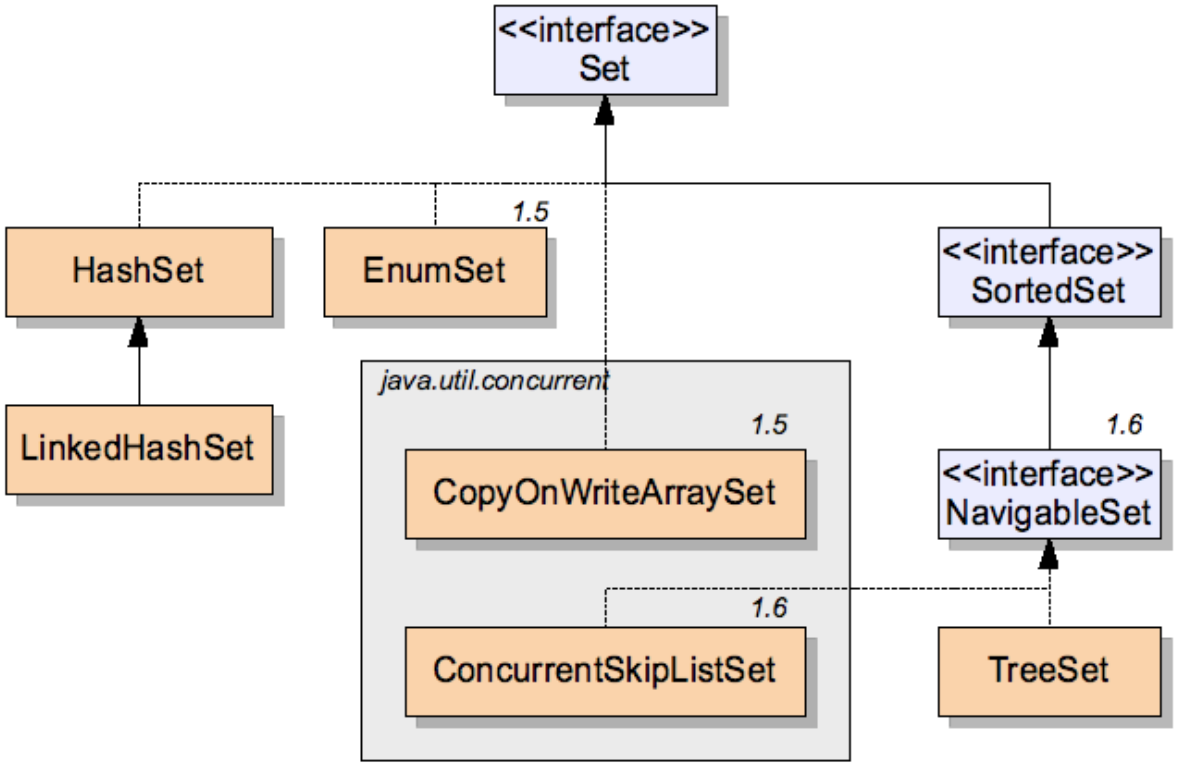
headSet(E toElem) : SortedSet<E>
subSet(E fromElem, E toElem) : SortedSet<E>
tailSet(E fromElem) : SortedSet<E>

comparator() : Comparator<? super E>

NavigableSet

pollFirst() : E
pollLast() : E
subSet(E from, boolean inclusive, E to,
    boolean inclusive) : NavigableSet<E>
headSet(E to, boolean inclusive) : NavigableSet<E>
tailSet(E from, boolean inclusive) : NavigableSet<E>
ceiling(E e) : E
floor(E e) : E
higher(E e) : E
lower(E e) : E
descendingSet() : NavigableSet<E>
descendingIterator() : Iterator<E>

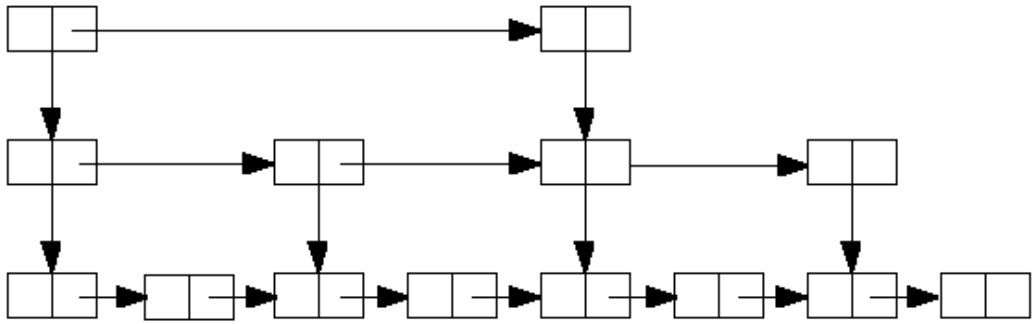
Set Implementations



Comparing Sets

	Data Structure	Sorting	Iterator	Nulls?
HashSet	Hash table	No	Fail-fast	Yes
Linked HashSet	Hash table + linked list	Insertion Order	Fail-fast	Yes
EnumSet	Bit vector	Natural Order	Weakly consistent	No
TreeSet	Red-black tree	Sorted	Fail-fast	Depends
CopyOnWrite ArraySet	Array	No	Snapshot	Yes
Concurrent SkipListSet	Skip list	Sorted	Weakly consistent	No

Skip Lists



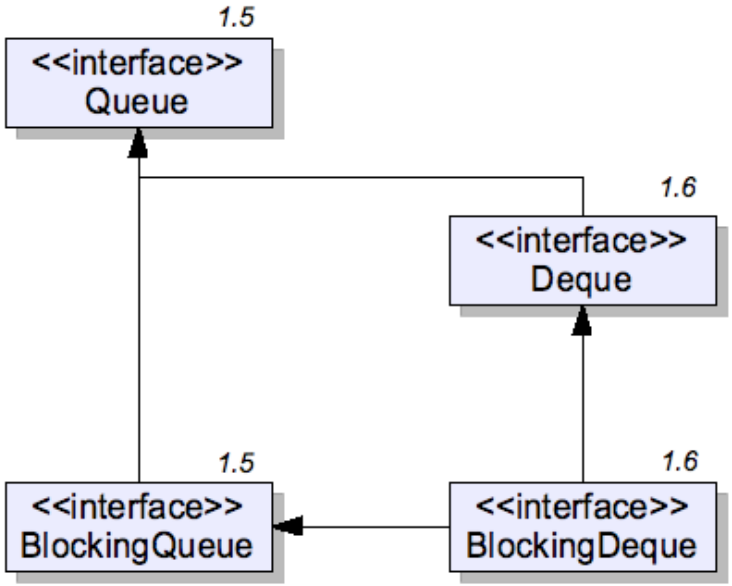
- Series of linked lists
- Reasonably fast search, add, remove
- Lock-free implementation!

Comparing Set Performance

	add	contains	next
HashSet	O(1)	O(1)	O(h/n)
Linked HashSet	O(1)	O(1)	O(1)
EnumSet	O(1)	O(1)	O(1)
TreeSet	O(log n)	O(log n)	O(log n)
CopyOnWrite ArraySet	O(n)	O(n)	O(1)
Concurrent SkipListSet	O(log n)	O(log n)	O(1)

Demo

Queues and Deques



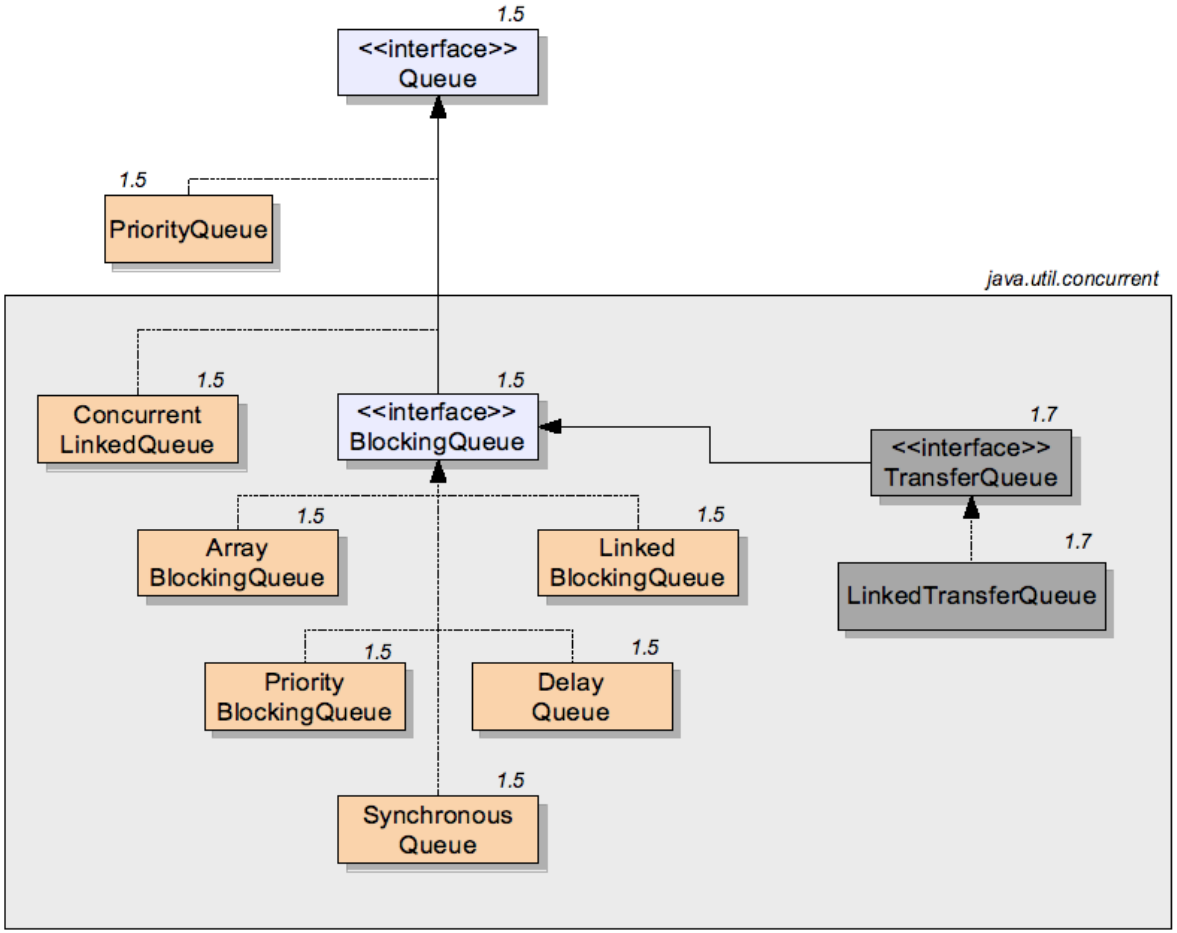
Queue Methods

	Throws exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

BlockingQueue Methods

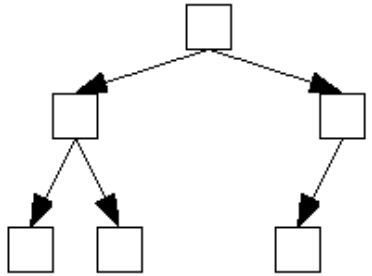
	Throws exception	Returns special	Blocks	Times out
Insert	add(E)	offer(E)	put(e)	offer(e, time, unit)
Remove	remove()	poll()	take()	poll(time, unit)
Examine	element()	peek()	X	X

Queue Implementations



Demo

Data Structure: Priority Heap



- Balanced binary tree
- Root is always highest priority
- Inserts and removes cause re-balancing

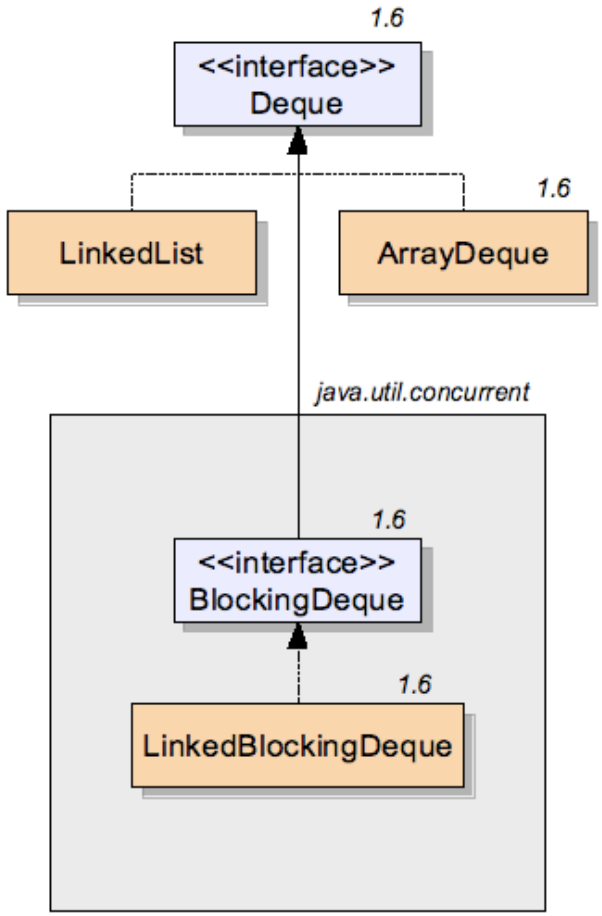
Deque Methods

	Head: Throws exception	Head: Special value	Tail: Throws exception	Tail: Special value
Insert	addFirst(e) *Stack: push*	offerFirst(e)	addLast(e) *Queue: add*	offerLast(e) *Queue: offer*
Remove	removeFirst() *Queue: remove* *Stack: pop*	pollFirst() *Queue: poll*	removeLast()	pollLast()
Examine	getFirst() *Queue: element*	peekFirst() *Queue: peek* *Stack: peek*	getLast()	peekLast()

Blocking Deque Methods

HEAD:	Throws exception	Special value	Blocks	Times out
Insert	addFirst(e)	offerFirst(e)	putFirst(e)	offerFirst(e,time,unit)
Remove	removeFirst() *Queue: remove*	pollFirst() *Queue: poll*	takeFirst() *Queue: take*	pollFirst(time, unit) *Queue: poll*
Examine	getFirst() *Queue: element*	peekFirst() *Queue: peek*	X	X
TAIL:	Throws exception	Special value	Blocks	Times out
Insert	addLast(e) *Queue: add*	offerLast(e) *Queue: offer*	putLast(e) *Queue: put*	offerLast(e,time,unit) *Queue: offer*
Remove	removeLast()	pollLast()	takeLast()	pollLast(time, unit)
Examine	getLast()	peekLast()	X	X

Deque Implementations



Comparing Queue Implementations

	Data Structure	Sorting	Bounds	Nulls?
PriorityQueue	Priority heap	Sorted	Unbounded	No
ArrayDeque	Array	FIFO	Unbounded	No
LinkedList	Linked list	FIFO	Unbounded	Yes
ConcurrentLinkedQueue	Linked list	FIFO	Unbounded	No
ArrayBlockingQueue	Array	FIFO	Bounded	No
PriorityBlockingQueue	Priority heap	Sorted	Unbounded	No
SynchronousQueue	none!	N/A	0	No
DelayQueue	Priority heap	Delayed order	Unbounded	No
LinkedBlockingQueue	Linked list	FIFO	(Un)bounded	No
LinkedBlockingDeque	Linked list	FIFO	(Un)bounded	No

Queue Performance

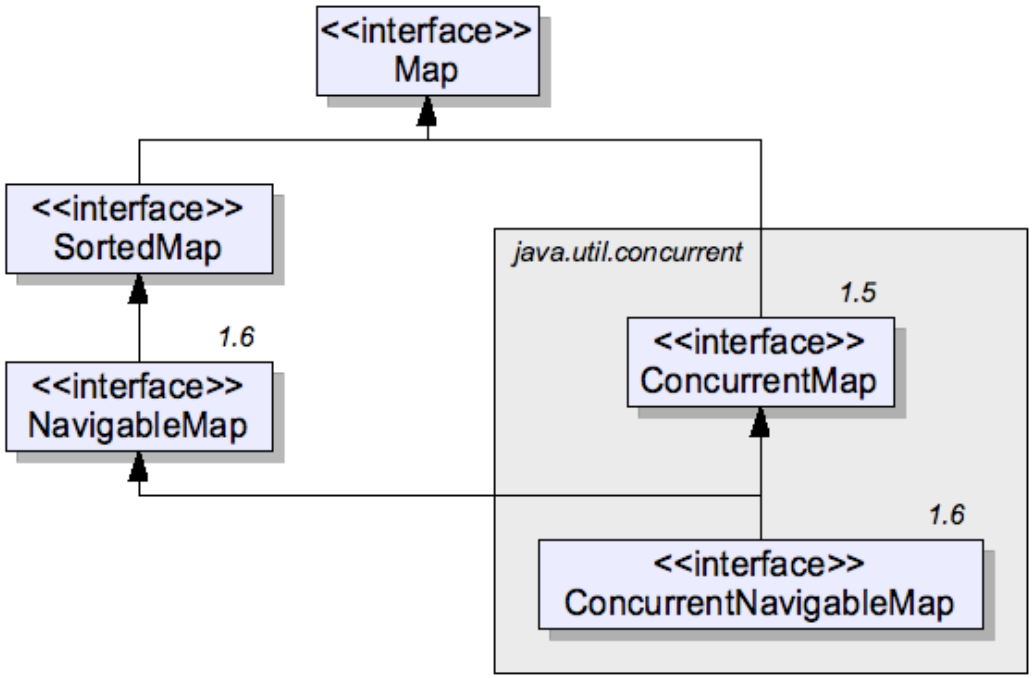
	offer	peek	poll	size
PriorityQueue	O(log n)	O(1)	O(log n)	O(1)
LinkedList	O(1)	O(1)	O(1)	O(1)
ArrayDeque	O(1)	O(1)	O(1)	O(1)
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(1)
PriorityBlockingQueue	O(log n)	O(1)	O(log n)	O(1)
SynchronousQueue	O(1)	O(1)	O(1)	O(1)
DelayQueue	O(log n)	O(1)	O(log n)	O(1)
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(1)
LinkedBlockingDeque	O(1)	O(1)	O(1)	O(1)

TransferQueue

Producers wait for consumers to receive elements. Useful for message passing. Similar to a broader version of SynchronousQueue.

hasWaitingConsumer() : boolean
getWaitingConsumerCount() : int
transfer(E e)
tryTransfer(E e) : boolean
tryTransfer(E e, long timeout, TimeUnit unit) : boolean

Map Interfaces



Map

put(E key,V value) :V
putAll(Map<? extends K, ? extends V> m)
remove(Object key) :V
clear()
containsKey(Object key) : boolean
containsValue(Object value) : boolean
isEmpty() : boolean
size() : int
get(Object key) :V
entrySet() : Set<Map.Entry<K,V>>
keySet() : Set<K>
values() : Collection<V>

SortedMap

firstKey() : K
lastKey() : K
headMap(K to) : SortedMap<K,V>
subMap(K from, K to) : SortedMap<K,V>
tailMap(K from) : SortedMap<K,V>
comparator() : Comparator<? super K>

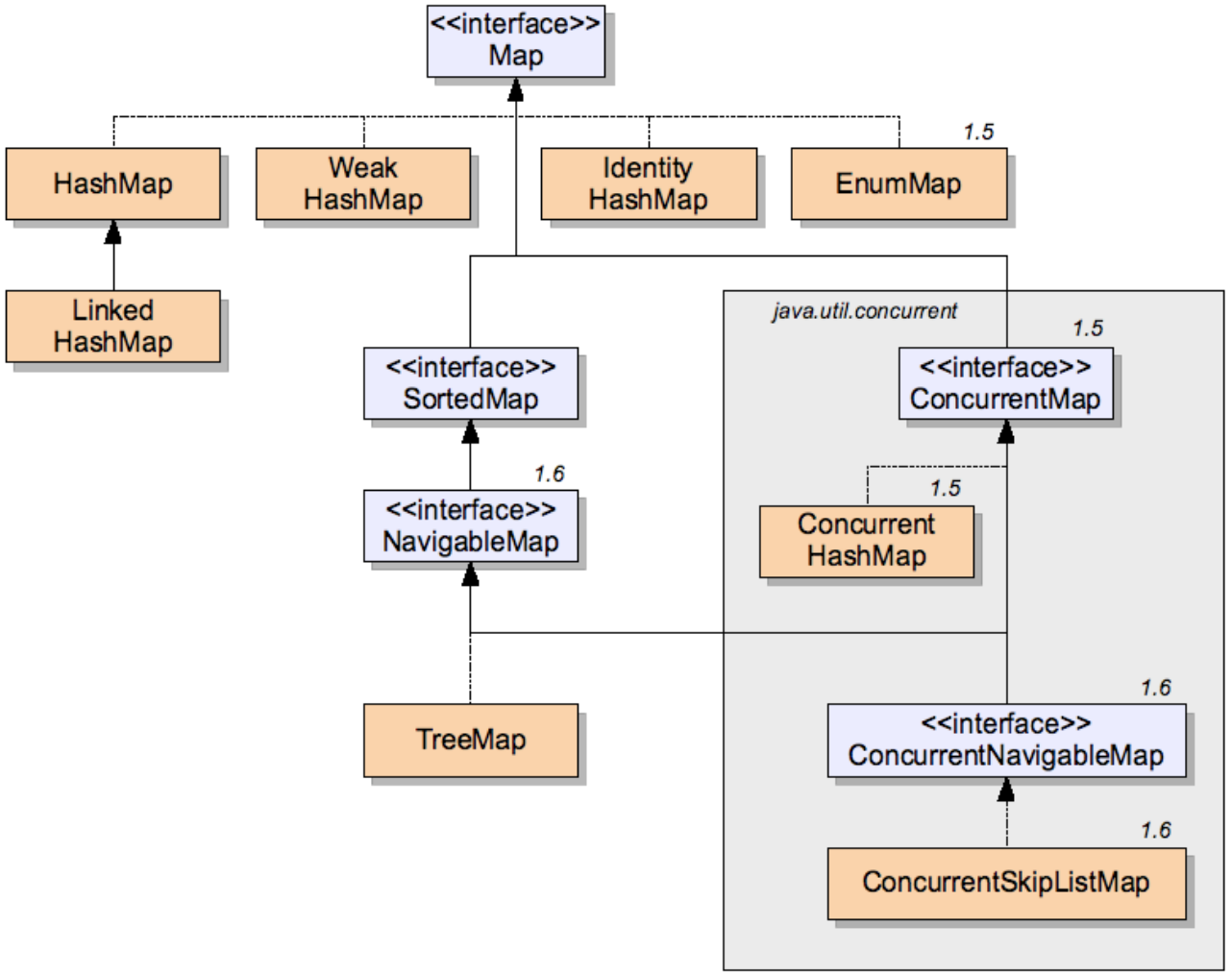
NavigableMap

firstEntry() : Map.Entry<K,V>
lastEntry() : Map.Entry<K,V>
ceilingEntry(K key) : Map.Entry<K,V>
ceilingKey(K key) : K
floorEntry(K key) : Map.Entry<K,V>
floorKey(K key) : K
higherEntry(K key) : Map.Entry<K,V>
higherKey(K key) : K
lowerEntry() : Map.Entry<K,V>
lowerEntry(K key) : K
navigableKeySet() : NavigableSet<K>
pollFirstEntry() : Map.Entry<K,V>
pollLastEntry() : Map.Entry<K,V>
headMap(K to, boolean inclusive) : NavigableMap<K,V>
subMap(K from, boolean fromInc, K to, boolean toInc) : NavigableMap<K,V>
tailMap(K from, boolean inclusive) : NavigableMap<K,V>
descendingKeySet() : NavigableSet<K>
descendingMap() : NavigableMap<K,V>

ConcurrentMap

putIfAbsent(K key,V value) :V
remove(Object key, Object value) : boolean
replace(K key,V value) :V
replace(K key,V oldValue,V newValue) : boolean

Map Implementations



Comparing Map Implementations

	Data Structure	Sorting	Iterator	Nulls?
HashMap	Hash table	No	Fail-fast	Yes
LinkedHashMap	Hash table + linked list	Insertion or access order	Fail-fast	Yes
IdentityHashMap	Array	No	Fail-fast	Yes
WeakHashMap	Hash table	No	Fail-fast	Yes
EnumMap	Array	Natural order	Weakly consistent	No
TreeMap	Red-black tree	Sorted	Fail-fast	Yes
ConcurrentHashMap	Hash tables	No	Weakly consistent	No
ConcurrentSkipListMap	Skip list	Sorted	Fail-fast	No

Map Performance

	get	containsKey	next
HashMap	O(1)	O(1)	O(h/n)
LinkedHashMap	O(1)	O(1)	O(1)
IdentityHashMap	O(1)	O(1)	O(h/n)
WeakHashMap	O(1)	O(1)	O(h/n)
EnumMap	O(1)	O(1)	O(1)
TreeMap	O(log n)	O(log n)	O(log n)
ConcurrentHashMap	O(1)	O(1)	O(h/n)
ConcurrentSkipListMap	O(log n)	O(log n)	O(1)

Demo

Collections

Collection algorithms:
- min
- max
- frequency
- disjoint

List algorithms:
- sort
- binarySearch
- reverse
- shuffle
- swap
- fill
- copy
- replaceAll
- indexOfSubList
- lastIndexOfSubList

Factories:
- EMPTY_SET
- EMPTY_LIST
- EMPTY_MAP
- emptySet
- emptyList
- emptyMap
- singleton
- singletonList
- singletonMap
- nCopies
- list(Enumeration)

Comparators:
- reverseOrder

Miscellaneous:
- addAll
- enumeration

Wrappers:
- unmodifiableCollection
- unmodifiableSet
- unmodifiableSortedSet
- unmodifiableList
- unmodifiableMap
- unmodifiableSortedMap
- synchronizedCollection
- synchronizedSet
- synchronizedSortedSet
- synchronizedList
- synchronizedMap
- synchronizedSortedMap
- checkedCollection
- checkedSet
- checkedSortedSet
- checkedList
- checkedMap
- checkedSortedMap

References

- Concurrency JSR 166 Interest Site, http://gee.cs.oswego.edu/dl/concurrency-interest/index.html
- Introduction to Algorithms, Cormen et al
- Java Concurrency in Practice, Goetz et al
- Java Generics and Collections, Naftalin and Wadler
- Java Platform API Specification, http://java.sun.com/javase/6/docs/api/
- Lecture on Skip Lists, Prof. Charles Leiserson, http://ocw.mit.edu/ans7870/6/6.046j/f05/lecturenotes/ocw-6.046-26oct2005.mp3

Bonus Slides

Other Collection Libraries

- Google Collections
- Apache Commons Collections
- High-Scale Library (Cliff Click)
- Gnu Trove
- Doug Lea's Collections

Google Collections

- http://code.google.com/p/google-collections/
- New types:
  - BiMap - bidirectional map
  - Multiset - set that allows dups
  - Multimap - like Map, but allows dup keys
- New implementations
  - ReferenceMap - concurrent, references

Google Collections

- Utilities:
  - Comparators - primitives, compound, function-based, etc
  - Iterators - cycle, skip, find, transform, concat
  - Helpers for standard Collection types, primitive arrays, etc
  - And much more...

Apache Commons Collections

- http://commons.apache.org/collections/
- New types:
  - Bags
  - Buffer
  - BidiMap
- Decorators: type checking, transformation
- Implementations: composites, identity map, reference map
- Comparators, iterators, adapters, etc

High-scale Lib

- http://sourceforge.net/projects/high-scale-lib
- Designed for non-blocking use in large core environments
- Non-blocking hash table, set
- From Cliff Click at Azul

Gnu Trove

- http://trove4j.sourceforge.net/
- High-performance Map implementations
  - Customizable hash strategies
  - Primitive-specific
- Stack, List, Set, etc
  - Primitive-specific
- Procedure-based iteration

Doug Lea's Collections

- http://gee.cs.oswego.edu/dl/classes/collections/index.html
- No longer supported