Useful Java Collections

Nathaniel Osgood
MIT 15.879

May 9, 2012
Java.util Collection Hierarchies

```java
from collections.abc import Collection, Iterable, Iterator

class Set(E):
    class SortedSet(E):
        class TreeSet(E):
            pass
        class LinkedHashSet(E):
            pass

    class HashSet(E):
        pass

    class EnumSet(E):
        pass

    class PriorityQueue(E):
        pass

    class List(E):
        class ArrayList(E):
            pass
        class LinkedList(E):
            pass

class Queue(E):
    class PriorityQueue(E):
        pass

class Map(K,V):
    class SortedMap(K,V):
        class TreeMap(K,V):
            pass
        class WeakHashMap(K,V):
            pass
        class HashMap(K,V):
            pass
        class EnumMap(K,V):
            pass
```

```text
Java.util Collection Hierarchies

```

```java
from collections.abc import Collection, Iterable, Iterator

class Set(E):
    class SortedSet(E):
        class TreeSet(E):
            pass
        class LinkedHashSet(E):
            pass

    class HashSet(E):
        pass

    class EnumSet(E):
        pass

    class PriorityQueue(E):
        pass

    class List(E):
        class ArrayList(E):
            pass
        class LinkedList(E):
            pass

class Queue(E):
    class PriorityQueue(E):
        pass

class Map(K,V):
    class SortedMap(K,V):
        class TreeMap(K,V):
            pass
        class WeakHashMap(K,V):
            pass
        class HashMap(K,V):
            pass
        class EnumMap(K,V):
            pass
```
Collection Objects in AnyLogic
Informal Collection Variables in AnyLogic
Useful Collections

• Array
• ArrayList
• Linked list
• Dictionary (e.g. implementation: HashTable)
• Set
• Priority queue
• Binary Tree
Common Characteristics

• Capacity to store information
• Iteration thru elements (support for Iterable interface)
• Separation of *interface* from *implementation*
  – There are often several particular “implementations” that can match a given interface (contract)
    • These differ in the details of “how” they accomplish the tasks prescribed in the implementation, but adhere to the contract
  – You can create your own implementation of an interface
• Java supports built-in rich set of collections
  – Java.collections
• Many collections use *generics* syntax to allow for “customized versions” for particular contents
  – e.g. ArrayList<**Person**>, HashMap<**String**, **Person**>
  – Note that these “generics” parameters *must be classes!*
Built-in Java Arrays

- Allocated via (explicit or implicit) new operator
- Can optionally list initial contents
- Can contain both “unboxed” (e.g. int) and “boxed” (e.g. Integer) contents
- Syntax Examples
  
  int arrayNeighborIndices[] = new int[n];
  String arrayCities[] = { “Bangor”, “Portland”, “Mooselookmeguntic” }; 

- Note that can have array of size 0
Java Arrays: Tradeoffs

• Pros
  – Can easily specify initial contents
  – Simple syntax
  – Boxed & unboxed elements
  – Fast lookup (by index)

• Cons
  – Painful to extend or delete elements (need to manually copy elements)
  – Only integer (int, short) indices
ArrayList

• Generic class
  – Syntax: ArrayList<Int>, ArrayList<String>

• Pros
  – Rapid insertion & deletion
  – Convenient integer-based indexing

• Note that can have empty ArrayList

• Combines good aspects of
  – Array
  – Linked list

• Suggestion: Use a built-in array if you know the size ahead of time
Linked List

• A sequential list of elements of arbitrary length
• Can iterate forward down list
• This is a Doubly Linked List
  – can iterate backwards in list (from end back to beginning)
• Be prepared for potential empty list!
• Application example: Linked List of History Information, *Persons* who have been infected, in order of infection occurrence
Dictionary (Hashtable and HashMap as Implementation)

- If key is the same as value, can be used to implement “content-indexed memory” (and “associative arrays”)
  - Cf:
    - Array: Look up content at integer
    - Dictionary: Can lookup many types of keys
      - e.g. look up information associated with String

- Example use of generics: HashMap<String, Person>, Hashtable<String, ArrayList<Person>>

- Two associated collections
  - Keys
  - Values
  - (each key can be used to look up an associated value)
Hashtable and HashMap Implementation of Map

• Pros
  – Rapid insertions (flexible size)
  – Can readily inset items by associated information

• Cons
  – Low bucket count => Risk of clashes between keys => longer time for insertions
    • If too few “buckets”, performance can grow similar to linked list
  – Larger data structure (“wasted space” in the form of empty buckets if load factor is off)

• Application Example: Look up City Characteristics for Names
Hashtable vs. HashMap

• Tolerance for null
  – Hashtable prohibits null keys & values
  – HashMap allows
    • One null key
    • Many null values

• HashMap has a subclass with a predictable order of iteration

• HashMap and Hashtable also differ with respect to multithreading support (beyond scope of course)
Set

- Dichotomous inclusion/omission
  - .contains
- No ordering of elements
  - Cannot tell if A was inserted before B or vice-versa
- Set operations
  - Union (.addAll())
  - Intersection (.retainAll())
  - (Assymmetric) Set difference (.removeAll())
- Example: Keeping track of *Persons* that have been infected thus far
Priority Queue

• For a given priority level, this is first-in-first-out
  – First inserted is first to reach “head” of queue

• Can prioritize according to arbitrary comparator
  – Like “first class” vs. “economy” lane, those with higher priority can “skip ahead” of those with lower priority

• A key use lies in representing a waiting list

• Getting element at head
  – Call to poll() (returns reference to element and removes from head of queue)
  – Call to peek() (returns reference to item at queue head)
Building Your Own Collections

• Java developers routinely create novel “data structures”, including some collections
• Often these data structures are composed of pieces using the above-described (“built-in”) collections
• If you build your own collections, be aware that care should be taken in several areas
  – Need to be careful about passing out references to values from the collection, in case they can be modified
  – Need to be careful about storing away references to external values, since this might allow external code to (typically, inadvertently) modify the data structure internals