SADI TuteLab 5

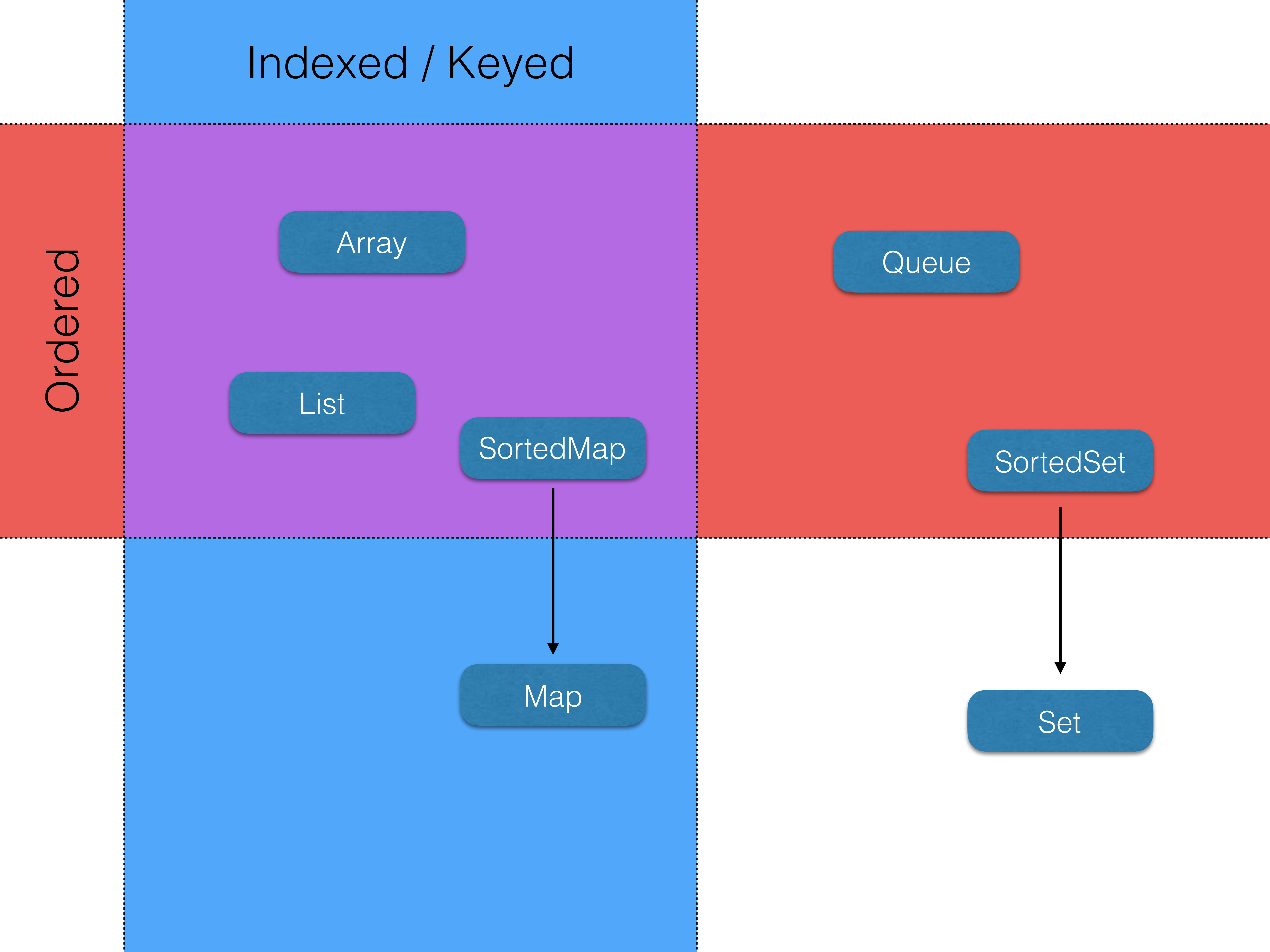
Java Collections Framework & Generics

Joshua Beale 2018

What are the main interfaces in Java Collection Framework (JCF)?

- Collection
    - Set
        - SortedSet
            - NavigableSet
    - List
    - Queue
        - Deque

- Map
    - SortedMap
        - NavigableMap


Indexed / Keyed
Ordered
Array
Queue
List
SortedMap
SortedSet
Map
Set

What is the advantage of using the Java Collection or Map classes (as opposed to building your own)?

- Why do extra work?
- There WILL be bugs
- I mean, do you really want to do MORE work for no reason?
- Interoperability with other packages
- I mean, WHY!?!??!!!??!?

- Customisation
  (but you could just write a wrapper class)
- (Theoretically) better performance
  (if you write a stripped down version)

- Examine the class/interface structure of the java.util.ArrayList<E> class in the Java API documentation and explain how abstract classes and interfaces have been used to facilitate reusability and extensibility.

- What about generics, how and why are they used?

ArrayList<E>

- `Collection<E>` **`extends`** `Iterable<E>`
  Defines add, contains, remove, clear, size & toArray methods
  Allows Collections to be handled as Collection type rather than specific collection type (i.e. List, Set)
  - `List<E>` **`extends`** `Collection<E>`
    Defines add (at index), get, set, remove (at index) & listIterator methods
    Allows Lists to be handled as List type rather than concrete subtype.

- `AbstractCollection<E>` **`implements`** `Collection<E>`
  Provides basic contains, toArray, remove & clear operations based on iterator().
  - `AbstractList<E>` **`extends`** `AbstractCollection<E>` **`implements`** `List<E>`
    Provides basic removeRange, iterator, listIterator & subList operations based on get() set(), add() and remove() (which are unimplemented)
    Basic List can be implemented with override of:
      get(int), size() (unmodifiable)
      set(int, E) (modifiable)
      add(int, E), remove(int) (variable size list)

    - `ArrayList<E>` **`extends`** `AbstractList<E>`
      `implements` `List<E>, RandomAccess …`
      Completely reimplements almost all inherited methods! (for efficiency)

ArrayList<E> vs ArrayList

- Generics allow compile time checking of add, set, sort, removeIf & forEach method parameters.

- Generics remove need for casting after get, remove(index) & set method calls or when using Iterator/ListIterator/Spliterator

```java
// Positional Access Operations

@SuppressWarnings("unchecked")
E elementData(int index) {
    return (E) elementData[index];
}
```

When is it more suitable to use a LinkedList over an ArrayList?

Operation	ArrayList	LinkedList
get(0), get(N-1), get(x)	O(1)	O(1), O(1), O(N/4)
add(0), add(N-1), add(x)	O(N), O(1) or O(N), O(N/2)	O(1), O(1), O(N/4)
remove(0), remove(N-1), remove(x)	O(N), O(1), O(N/2)	O(1), O(1), O(N/4)
Iterator.add/remove(0), Iterator(N-1), Iterator(x)	O(N), O(1), O(N/2)	O(1), O(1), O(1)

https://javatutorial.net/difference-between-arraylist-and-linkedlist-in-java https://twitter.com/joshbloch/status/583813919019573248

Why not ~~Zoidberg~~ ArrayDeque?
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayDeque.html

Does the sub-interface Set introduce any additional methods to Collection interface? Does it introduce any additional constraints?

Extra methods defined by Set<E>:

None…

Methods redeclared for javadoc purposes

Extra constraints introduced by Set<E>:

- no duplicate elements
- requires equals() and hashcode() of elements to follow standard rules

What is the difference between Map and Collection classes?
What are the constraints imposed by the Map interface?

Collection<E>	Map<K, V>
• Elements	• Key-Value pairs
• Duplicates okay (except Sets)	• No duplicate Keys
• add(E)	• put(K, V)
• contains(E)	• containsKey(K)
	• containsValue(V)

Why does the Vector class provide methods such as addElement(Object o) in addition to add(Object o) defined in the interface Collection?

- Vector JavaDoc:
  Since: JDK1.0

- Collection/List JavaDoc:
  Since: JDK1.2

- As of the Java 2 platform v1.2, this class was retrofitted to implement the List interface, making it a member of the Java Collections Framework.

- Older methods were left in for backwards compatibility

Which collection(s) will be suitable for:

- List of Goods received. New entries to be added at the end reflecting the date and time of goods receipt.
- A text editor storing each line as an element in a list. Users are allowed to add, delete or alter any part of the document.

List of Goods received. New entries to be added at the end reflecting the date and time of goods receipt.

- ArrayList is a solid first choice:
  - New elements are added to the end
  - Minimal removals likely
- A Queue might be an option if random access is not important:
  - ArrayDeque
  - PriorityQueue if ordering is important and entries may be added in an incorrect order

A text editor storing each line as an element in a list. Users are allowed to add, delete or alter any part of the document.

- ArrayList may be suitable
  - may suffer from insertion/deletion partway through the document
- A Queue
  - probably impractical due to lack of random access methods
- LinkedList is the only real JCF List alternative
  - Iterator should be used where possible
- To be honest, a text editor usually spends most of it’s time waiting for the user to do something so processing efficiency is not the most important factor most of the time