Today …

• Quiz 7

• The Java Collection Framework
  – Exercise 13

• Reminder
  – Next Thursday your group projects are due!

• Reading Assignment
  – Core: Ch. 13: 649-714 (Collection Framework)
The Java Collection Framework

- A set of interfaces and classes
  - Within java.util and java.util.concurrent
- APIs for common abstract data types (ADTs)
- Various implementations of each ADT
  - Recall that most ADTs have many implementations
  - Each having different trade-offs
    (asymptotic complexity of different operations)

Main interfaces

```
Iterable<T>

Collection<E>

List<E>  Set<E>  Queue<E>  e.g., FIFO

Duplicates + order

No duplicates, order not guaranteed

BlockingQueue<E>  Deque<E>

Waits for ops to complete (e.g., until space free to add)

BlockingDeque<E>

* E is a type variable standing for a collection “element”
```
A *Map* data type is a hash table *abstraction*
- Associates values (of type V) to keys (of type K)
- Values can be accessed using their keys (e.g., SSN → Person)
- Keys are assumed unique (so SSN maps to only one Person)
- Does not inherit from Iterable or Collection

The basic Collection API

- **Adding elements**
  ```java
  boolean add(E elt)  // Why use the extends wildcard here?
  boolean addAll(Collection<? extends E> coll)
  ```

- **Removing elements**
  ```java
  boolean remove(Object o)  // Short for:
  void clear()  // T> boolean removeAll(collection<T> coll)
  boolean removeAll(Collection<?> coll)
  boolean retainAll(Collection<?> coll)
  ```
The basic Collection API

• Querying the Contents of a Collection

  boolean contains(Object elt)  
  boolean containsAll(Collection<?> coll)
  boolean isEmpty()
  int size()

• Other ways to access collection elements

  Iterator<E> iterator()
  Object[] toArray()  // for legacy applications
  <T> T[] toArray(T[] t)  // copies into t

Can result in runtime errors

Sets

• Adding an element to a set only has an effect …
  – If the element is not already in the set
  – “already in the set” is determined using equals()

• Different implementations

  – HashSet
    • based on a hash table
    • most common
  – LinkedHashSet
    • Maintains a linked list of elements
    • E.g., to traverse in insertion order
  – TreeSet
    • Based on TreeMap, a Red-Black (balanced) binary tree implementation
Set Example
Set<String> s1 = new HashSet<String>();
s1.add("Hello");
s1.add("Hello"); // what does this do?
Set<String> s2 = new LinkedHashSet<String>();
s2.add("Hello");
s2.add("World");
Set<Integer> s3 = new TreeSet<Integer>();
s3.add(1);
s3.add(2);

Queues
• Generally First-In First-Out (FIFO)
  – But can use other rules can be used as well (like task priority)
• Adds more methods to Collection
  Add an element to the end of the queue
  boolean offer(E elt) // may return false if bounded
  Throw exceptions for empty queues
  E element() // retrieve start but don’t remove
  E remove() // remove start and return
  Return null for empty queues
  E peek() // retrieve start but don’t remove
  E poll() // remove start and return
Queues

• There are various implementations
  – Each has different features

• Examples:
  – LinkedList
  – PriorityQueue (based on a heap)
  – ArrayDeque
  – ConcurrentLinkedQueue
  – And so on

The Collections Class

• A set of static helper methods for collections
  – We’ll go over some of these
  – Good idea to look through these methods!

• Many methods use the Comparator interface

  public interface Comparator<T> {
    int compare(T o1, T o2);
  }

  – Compare returns: <0 if o1<o2, 0 if o1=o2, and >0 if o1>o2
  – Many classes implement comparator
  – You can also pass a comparator object into methods
The Collections Class

Changing the order of collection elements

void reverse(List<?> list)  // reverse order

void shuffle(List<?> list)  // randomly permute

// sort a list of comparables
<T extends Comparable<? super T>> void sort(List<T> list)

// sort a list given a comparator
<T> void sort(List<T> list, Comparator<? super T> c)

// swap two indexes within a list
void swap(List<?> list, int i, int j)

The Collections Class Examples

List<String> list = Arrays.asList("a", "ab", "aab");

// sort it alphabetically
Collections.sort(list);
assert list.toString().equals("[a, aab, ab]");

// reverse it
Collections.reverse(list);
assert list.toString().equals("[ab, aab, a]");

// sort it from largest to smallest
Collections.sort(list, new Comparator<String>() {
    public int compare(String o1, String o2) {
        return o2.length() - o1.length();
    }
});
assert list.toString().equals("[aab, ab, a]");
The Collections Class

Changing the contents of lists

<T> void copy(List<? super T> dst, List<? extends T> src)
<T> void fill(List<? super T> list, T obj)
<T> boolean replaceAll(List<T> list, T oldVal, T newVal)

Finding specific values in a list

<T> int binarySearch(
    List<? extends Comparable<? super T>> list, T key)
int indexOfSubList(List<?> source, List<?> target)

Others

boolean disjoint(Collection<?> c1, Collection<?> c2)
int frequency(Collection<?> c, Object o)

Exercise 13, Part 1