Collections

The Collections Framework
forEach
Ordering Collections
The Collections Framework

• Interfaces
  – Implementation-agnostic interfaces for collections

• Implementations
  – Concrete implementations

• Algorithms
  – Searching, sorting, etc.

Using the framework saves writing your own: better performance, fewer bugs, less work, etc.
The Collection Interface

- Basic operators
  - size, isEmpty(), contains(), add(), remove()

- Traversal
  - for-each, and iterators

- Bulk operators
  - containsAll(), addAll(), removeAll(), retainAll(), clear()

- Array operators
  - convert to and from arrays
Collection Types

• Primary collection types:
  – Set (no duplicates, mathematical set)
  – List (ordered elements)
  – Queue (shared work queues)
  – Map (<key, value> pairs)

• Each collection type is defined as an interface
  – You need to choose a concrete collection
  – Your choice will depend on your needs
## Concrete Collection Types

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Hash table</th>
<th>Resizable array</th>
<th>Tree</th>
<th>Linked list</th>
<th>Hash table + linked list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set</strong></td>
<td>HashSet</td>
<td></td>
<td>TreeSet</td>
<td></td>
<td>LinkedHashSet</td>
</tr>
<tr>
<td><strong>List</strong></td>
<td></td>
<td>ArrayList</td>
<td></td>
<td>LinkedList</td>
<td></td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td></td>
<td>ArrayDeque</td>
<td></td>
<td>LinkedList</td>
<td></td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td>HashMap</td>
<td></td>
<td>TreeMap</td>
<td></td>
<td>LinkedHashMap</td>
</tr>
</tbody>
</table>

Based on table from http://docs.oracle.com/javase/tutorial/collections/implementations/index.html
Four Commonly Used Collection Types

- **HashSet** implements a set as a hash table
  - Makes no ordering guarantees
- **ArrayList** implements a list using an array
  - Very fast access
- **HashMap** implements a map using a hash table
  - Makes no ordering guarantees
- **LinkedList** implements a queue using a linked list
  - First-in-first-out (FIFO) queue ordering
forEach

- Collections implement the `forEach` method, which applies an action to every element in the collection.

  Instead of:

  ```java
  for (Thing t : things) {
    System.out.println(t);
  }
  ```

  You can do this:

  ```java
  things.forEach(t -> System.out.println(t));
  ```
Ordering Collections

• The Comparable interface defines a ‘natural’ ordering for all instances of a given type, T:

```java
public interface Comparable<T> {
    public int compareTo(T o);
}
```

The return value is either negative, 0, or positive depending if the receiver comes before, equal, or after the argument, o.

• The Comparator interface allows a type T to be ordered in additional ways:

```java
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```
Collections.sort()

• No arguments
  – uses *natural* order for type

• Single Lambda argument:
  – uses order defined by lambda expression

  (a T, b T) -> { return <expression>; }
Josh Bloch Item 25: Prefer lists to arrays

• Why?
  – Arrays are covariant, Generics are invariant
    • if \( A \) extends \( B \), then \( A[] \) is a subclass of \( B[] \)
    • but \( \text{List}<A> \) has no relationship to \( \text{List}<B> \)

// Fails at runtime!
Object[] objectArray = new Long[1];
objectArray[0] = "I don’t fit in"; // Throws ArrayStoreException

// Won’t compile!
List<Object> ol = new ArrayList<Long>(); // Incompatible types
ol.add("I don’t fit in");