What is a Collection

- Object that groups multiple elements into a single unit
- Also called a “container”
- Used to store, receive, and manipulate data, and to transmit data from one method to another
- Represent data items that form a natural group (e.g., mail folder)

Collections in Earlier Versions of Java

- Included Vectors, Hashtable, and Array
- Did not include collections framework
Collections Framework

- Unified architecture for representing and manipulating collections
- Collections frameworks include
  - Interfaces
    - ADTs representing collections
  - Implementations
    - Reusable data structures implementing the collection interfaces
  - Algorithms
    - Polymorphic/reusable methods performing computations

Benefit

- Reduces programming effort
- Increases program speed and quality
- Allows interoperability among APIs (e.g., node names v.s. column headings)
- Reduces the effort to learn/use new APIs taking collections as input/output
- Reduces effort to design new APIs
- Fosters software reuse
Design Goals

- API reasonably small in size and “conceptual weight”
  - Should not look alien to current Java programmer
  - Augment current facilities rather than replacing them
- All reasonable representations of collections should interoperate well
  - e.g., arrays viewable as collections, maps viewable as collections, etc.

Core Collection Interfaces

- Used to manipulate collections and pass them from one method to another
- Allows collections to be manipulated independently of the details of their representation
- Form a hierarchy

Core Collection Interfaces (continued)

```
```

```
```
Core Collection Interfaces (continued)
• JDK does not provide separate interfaces for each variant of each collection type
• Variants include: immutable, fixed-size, and append-only
• Modification operations in each interface are “optional”
• Unsupported operations throw “UnsupportedOperationException”

Collection
• Root of the collection hierarchy (most general least common denominator)
• Represents a group of objects, known as its elements
• Some collections allow duplicate elements
• Some collections are ordered
• No direct implementation in JDK
• More specific subinterfaces implemented in JDK (e.g., Set, List)

Set
• Cannot contain duplicate elements
• Models the mathematical set abstraction
• E.g., set of processes running on a given machine
<table>
<thead>
<tr>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Ordered collection (also called “sequence”)</td>
</tr>
<tr>
<td>• Can contain duplicate elements</td>
</tr>
<tr>
<td>• Precise control over where each element is inserted</td>
</tr>
<tr>
<td>• Elements can be accessed by their integer index (position)</td>
</tr>
<tr>
<td>• Relates to Vector in older version of Java</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Object that maps keys to values</td>
</tr>
<tr>
<td>• Cannot contain duplicate keys</td>
</tr>
<tr>
<td>• Each key can map to at most one value</td>
</tr>
<tr>
<td>• Relates to Hashtable in older version of Java</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Two ways to order objects:</td>
</tr>
<tr>
<td>– Comparable interface</td>
</tr>
<tr>
<td>• automatic natural order</td>
</tr>
<tr>
<td>– Comparator interface</td>
</tr>
<tr>
<td>• complete control over object ordering</td>
</tr>
<tr>
<td>• Comparable, and Comparator interfaces are part of the underlying infrastructure</td>
</tr>
</tbody>
</table>
Sorted Set

- Set that maintains its elements in ascending order
- Additional operations provided to take advantage of the ordering
- Used for word lists, membership rolls, etc.

Sorted Map

- Map that maintains its mappings in ascending key order
- Used for apps like dictionaries, telephone directories, etc.

Implementations

- Actual data objects used to store collections
- Implement the core collection interfaces
- Three kinds of implementations
  - General-purpose implementations
    - public classes providing primary implementations
  - Wrapper implementations
    - used with other implementations to provide added functionality
  - Convenience Implementations
    - efficient alternatives (e.g., singleton set)
General-Purpose Implementations

• HashSet
  – Hash table implementation of the Set interface
• TreeSet
  – Red-black tree implementation of the SortedSet interface
• ArrayList
  – Resizable-array implementation of the List interface (unsynchronized vector)

General-Purpose Implementations (continued)

• LinkedList
  – Doubly-linked list implementation of the List interface
  – Useful for queues and double-ended queues
• HashMap
  – Hash table implementation of the Map interface (unsynchronized Hashtable, null keys & values)
• TreeMap
  – Red-black tree implementation of the SortedMap interface

Wrapper Implementations

• Accessed solely via static factory methods
• Collections.unmodifiableInterface
  – Returns an unmodifiable view of a specified collection
  – Throws an exception if the user attempts to modify it
• Collections.synchronizedInterface
  – Guaranteed thread-safety as long as all the accesses to the backing collection are through the returned collection
### Convenience Implementations

- `Arrays.asList`
- `EMPTY SET` and `EMPTY LIST`
  - constants representing empty set/list
- `singleton`
  - Returns immutable “singleton” set containing only the specified object
- `nCopies`
  - Returns immutable list containing n copies of a specified object

### Legacy Implementations

- `Vector`
  - synchronized resizable-array implementation of the List interface with additional “legacy methods”
- `Hashtable`
  - synchronized hash table implementation of the Map interface that does not allow null keys or values, with additional “legacy methods”

### Special Purpose Implementations

- `WeakHashMap`
  - Implementation of the Map interface that only stores weak references to its keys
  - This allows key-value pairs to be garbage-collected when the key is no longer referenced outside of the WeakHashMap
  - Used to implement “registry-like” data structures
    - utility of an entry vanishes when its key is no longer reachable by any thread
Abstract Implementations

• JDK’s abstract implementations can be used to build custom implementations
• Abstract Collection (e.g., bag)
• Abstract Set
• Abstract List (e.g., array)
• AbstractSequentialList (e.g., linked list)
• Abstract Map (skeletal implementation of a map)

Algorithms

• Polymorphic algorithms
  – Pieces of reusable functionality provided by the JDK
  – All of them come from the Collections class
  – Static methods whose first argument is the collection on which to perform the operation
• Majority of algorithms operate on List objects
• A few operate on arbitrary Collections

Algorithms (continued)

• Sorting
  – Reorders a List so that its elements are ascending order according to some ordering relation
  – Uses an optimized merge sort algorithm
• Shuffling
  – Opposite of what sort does
• Routine Data Manipulation
  – reverse, fill, copy
Algorithms (continued)

• Searching
  – binarySearch algorithm (2 forms)
    • Takes a sorted List and an element to search
    • Takes a Comparator/List/element to search

• Finding Extreme Values (2 forms)
  – min and max algorithms
    • Return the min/max using natural ordering
    • Returns the min/max using a Comparator ordering

Algorithms Summary

• sort(List)
• binarySearch(List, Object)
• reverse(List)
• shuffle(List)
• fill(List, Object)
• copy(List dest, List src)
• min(Collection)
• max(Collection)

Custom Implementations

• Persistent
  – Veneer over an external database
  – Concurrent access by multiple VMs

• Application-specific
  – E.g., unmodifiable Map containing real-time telemetry data
    • keys are locations, values are sensor readings

• Highly Concurrent
  – Synchronization wrappers lock the collection
Custom Implementations (continued)

- High-performance, special-purpose
  - e.g., small fixed set using a bit-vector
- High-performance, general-purpose
  - Could come up with something faster
- Enhanced Functionality
  - e.g., bag implementation (multiset), use a hashMap to support constant time access and duplicates
- Convenience, or Adapter to legacy API

Infrastructure

- Iterators
  - Similar to Enumeration interface but more powerful
    - Iterator
      - allows removal of elements
    - ListIterator
      - supports bi-directional iteration, element replacement, element insertion and index retrieval
- Ordering
  - Comparable, Comparator

Infrastructure (continued)

- Runtime Exceptions
  - UnsupportedOperationException
  - ConcurrentModificationException
Array Utilities

• Arrays
  – Contain static methods to sort, search, compare, and fill arrays of primitives and Objects

Interoperability

• Compatibility
  – How collections can be made to work with older APIs
• API design
  – How to design new APIs that interoperate seamlessly with one another

Collections Framework v.s. STL

• How does the Java collections compare to STL in C++?
  – Somewhat similar to STL but also markedly different!
  – Iterators and iterator-pairs play a central role in STL
  – Collection interfaces play a central role in the JDK
  – JDK names have been chosen for clarity, brevity, and similarity