CSE 1325 - Object-Oriented Programming

Collections

Alex Dillhoff

University of Texas at Arlington

Collections

`Collections` in Java were not part of the original release.

The `Collections` framework provides many useful data structures and algorithms.

Collections

They include iterators which allow easy access to the underlying data.

Additionally, they are easily extensible. Custom collections can be created through subclassing.

Collection Interfaces

The base `Collection` interface is implemented by all collections.

It includes many common operations that any data structure can use.

Each method is adapted to fit the particular implementation (array, hash map, etc.).

Collection Interfaces

The following interfaces are extensions of the `Collection` interface.

There are many more than are listed in these slides. Refer to the official Java API for more information.

`List` Interface

The `List` interface is used for collections which are sequences of elements.

Each element is given a position in that sequence and is identified by that position.

Lists **may** contain duplicates.

List Interface

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/List.html

`ArrayList` Class

The `ArrayList` class implements the `List` interface.

It is useful because it supports dynamic arrays.

ArrayList Class

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/ArrayList.html

`LinkedList` Class

The `LinkedList` class implements the `List`, `Queue`, and `Deque` interfaces.

It provides an efficient linked list data structure.

LinkedList Class

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/LinkedList.html

Set Interface

The Set interface also defines a sequence of elements.

However, it does **not** allow duplicate elements.

Set Class

Documentation

`https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Set.html`

HashSet Class

The `HashSet` class implements `Set`.

Elements inserted into this data structure are placed into a hash table.

The hash code is set automatically.

HashSet Class

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/HashSet.html

TreeSet Class

The `TreeSet` class extends `AbstractSet` and implements `NavigableSet`.

Elements added to this must implement `Comparable`.

TreeSet Class

Elements are stored in sorted, ascending order.

This data structure is beneficial because it provides guaranteed $log(n)$ time for `add`, `remove`, and `contains`.

TreeSet Class

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/TreeSet.html

SortedSet Interface

The `SortedSet` interface extends `Set`.

Elements are sorted by **ascending** order when the set is created.

The elements must implement `Comparable`.

SortedSet Interface

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/SortedSet.html

NavigableSet Interface

The `NavigableSet` interface extends `SortedSet`.

It provides a way to retrieve items that are close to the search query.

For example, `higher(E e)` returns the least element in the set strictly greater than the given element, or `null` if there is no such element.

NavigableSet Interface

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/NavigableSet.html

Queue Interface

`Queue` declares methods required to implement a standard First In First Out (FIFO) queue.

Documentation

`https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Queue.html`

PriorityQueue Class

The `PriorityQueue` class orders element according to their natural ordering or by a `Comparator` provided to the constructor.

The *head* of the queue is the least element with respect to ordering.

Natural ordering refers to the object's implementation of `Comparable`.

PriorityQueue Class

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/PriorityQueue.html

Map Interface

`Maps` are key/value stores that do not allow for iterating.

However, you can obtain a `Collection` view of the map to access iterators.

Map Interface

Maps map unique keys to values. 1em
They are extremely efficient when adding, removing, and accessing objects.

Map Interface

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Map.html

HashMap Class

This class implements the `Map` interface.

It provides an optimized version of a hash table similar to the one you may have implemented in CSE 1320.

HashMap Class

Documentation

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/HashMap.html

Iterators

Iterators are used to iterate though a collection.

They do so by providing methods such as `hasNext()` and `next()`.

Iterators

A regular `Iterator`, returned using `iterator()`, allows you to traverse forward through a collection.

A `ListIterator`, returned using `listIterator()`, allows traversals in both directions as well as the ability to modify individual elements.

Iterators

Documentation

`https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Iterator.html`