Lecture 27

~~Dictionaries~~
TABLES

- Dictionaries
  - Terminology
  - Implementation
  - `structure` Package

Eddie Munster and the Addams Family

Driver's Ed' Full Sketch - I Think You Should Leave Season 2 aka **TABLES**

~~Dictionaries~~

TABLES

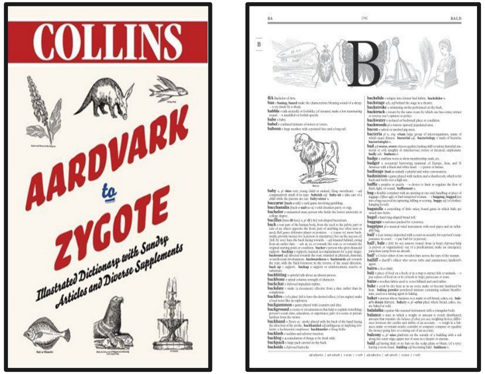
Dictionaries

An English dictionary is a map from words to their definitions.

However, it also has some additional features.

For example, we can ask what the “next” word in a dictionary is.

This is not possible in an arbitrary map


COLLINS
AARDVARK to ZYGOTE
Illustrated Dictionary with Sundry Articles and Diverse Supplements
B

Another English dictionary.

keys	values
	Ephs
	Trailblazers
	Llamas
	Falcons

A map from images to strings.

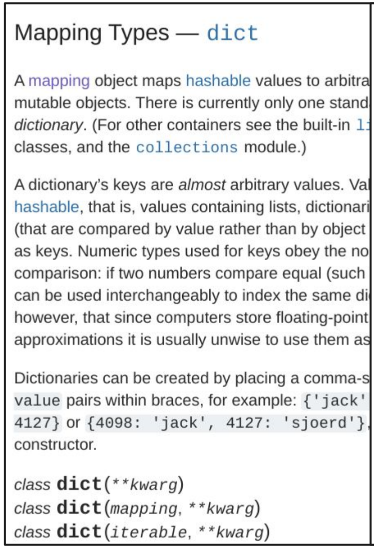
Do the images have an obvious order?

We model this in data structures by defining a *dictionary* to be a map with an additional property:

The keys are comparable (and hence orderable).

Note: The textbook and structure package use the term *table* instead of dictionary.

Terminology



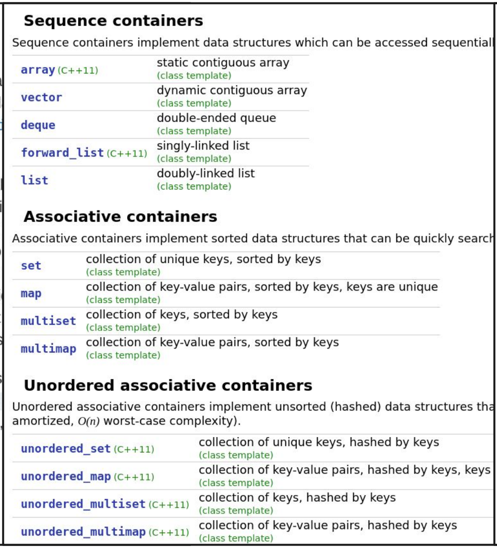
Mapping Types — `dict`

A mapping object maps hashable values to arbitra
mutable objects. There is currently only one stand
dictionary. (For other containers see the built-in `li`
classes, and the `collections` module.)

A dictionary's keys are *almost* arbitrary values. Val
hashable, that is, values containing lists, dictionari
(that are compared by value rather than by object
as keys. Numeric types used for keys obey the no
comparison: if two numbers compare equal (such
can be used interchangeably to index the same di
however, that since computers store floating-point
approximations it is usually unwise to use them as

Dictionaries can be created by placing a comma-s
`value` pairs within braces, for example: `{'jack'`
`4127}` or `{4098: 'jack', 4127: 'sjoerd'}`
constructor.

class **dict**(***kwarg*)
class **dict**(*mapping*, ***kwarg*)
class **dict**(*iterable*, ***kwarg*)



Sequence containers

Sequence containers implement data structures which can be accessed sequentiall

`array` (C++11)	static contiguous array (class template)
`vector`	dynamic contiguous array (class template)
`deque`	double-ended queue (class template)
`forward_list` (C++11)	singly-linked list (class template)
`list`	doubly-linked list (class template)

Associative containers

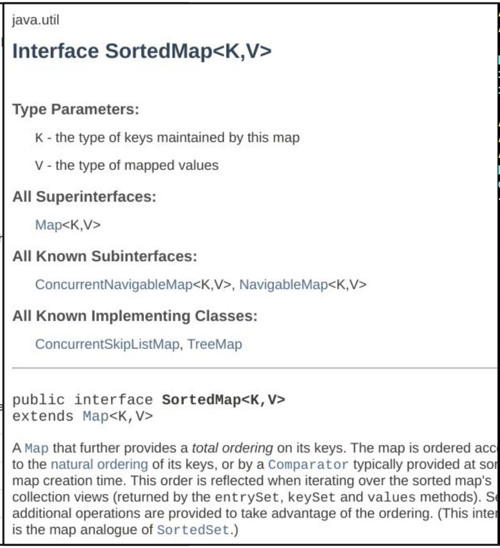
Associative containers implement sorted data structures that can be quickly search

`set`	collection of unique keys, sorted by keys (class template)
`map`	collection of key-value pairs, sorted by keys, keys are unique (class template)
`multiset`	collection of keys, sorted by keys (class template)
`multimap`	collection of key-value pairs, sorted by keys (class template)

Unordered associative containers

Unordered associative containers implement unsorted (hashed) data structures tha
amortized, $O(n)$ worst-case complexity).

`unordered_set` (C++11)	collection of unique keys, hashed by keys (class template)
`unordered_map` (C++11)	collection of key-value pairs, hashed by keys, keys (class template)
`unordered_multiset` (C++11)	collection of keys, hashed by keys (class template)
`unordered_multimap` (C++11)	collection of key-value pairs, hashed by keys (class template)



java.util

Interface SortedMap<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Superinterfaces:

Map<K,V>

All Known Subinterfaces:

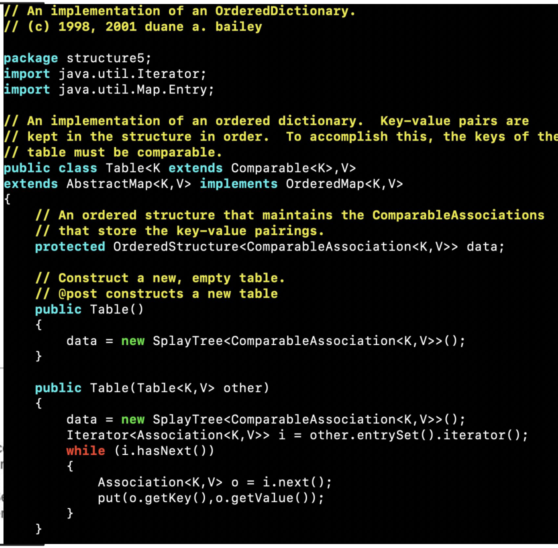
ConcurrentNavigableMap<K,V>, NavigableMap<K,V>

All Known Implementing Classes:

ConcurrentSkipListMap, TreeMap

```
public interface SortedMap<K,V>
extends Map<K,V>
```

A `Map` that further provides a *total ordering* on its keys. The map is ordered acc
to the natural ordering of its keys, or by a `Comparator` typically provided at so
map creation time. This order is reflected when iterating over the sorted map's
collection views (returned by the `entrySet`, `keySet` and `values` methods). Se
additional operations are provided to take advantage of the ordering. (This inte
is the map analogue of `SortedSet`.)



```
// An implementation of an OrderedDictionary.
// (c) 1998, 2001 duane a. bailey

package structure5;
import java.util.Iterator;
import java.util.Map.Entry;

// An implementation of an ordered dictionary.  Key-value pairs are
// kept in the structure in order.  To accomplish this, the keys of the
// table must be comparable.
public class Table<K extends Comparable<K>,V>
extends AbstractMap<K,V> implements OrderedMap<K,V>
{
    // An ordered structure that maintains the ComparableAssociations
    // that store the key-value pairings.
    protected OrderedStructure<ComparableAssociation<K,V>> data;

    // Construct a new, empty table.
    // @post constructs a new table
    public Table()
    {
        data = new SplayTree<ComparableAssociation<K,V>>();
    }

    public Table(Table<K,V> other)
    {
        data = new SplayTree<ComparableAssociation<K,V>>();
        Iterator<Association<K,V>> i = other.entrySet().iterator();
        while (i.hasNext())
        {
            Association<K,V> o = i.next();
            put(o.getKey(),o.getValue());
        }
    }
```

Note that the "map" and "dictionary" terminology is not standard across computer science.

- In Python, a `dict` is a mapping with hashable keys, and `map` applies a function to an iterable. Hashable implies orderable, so this aligns closely with the our use of dictionary in these slides.
- In the C++ standard library, a `map` has ordered keys (i.e., a dictionary here), and no dictionary.
- In Java's standard `java.util` package, `Map` is an `interface` for a map, and `SortedMap` is an `interface` for a map with ordered keys (i.e., a dictionary here).
- In the textbook and `structure` package, `Map` is an interface for a map, and `Table` is an `interface` for a map with ordered keys (i.e., a dictionary here).
- Wikipedia uses *associative array* for map, and *ordered dictionary* for ordered keys (i.e., a dictionary here).

Implementation

Implementing a Dictionary

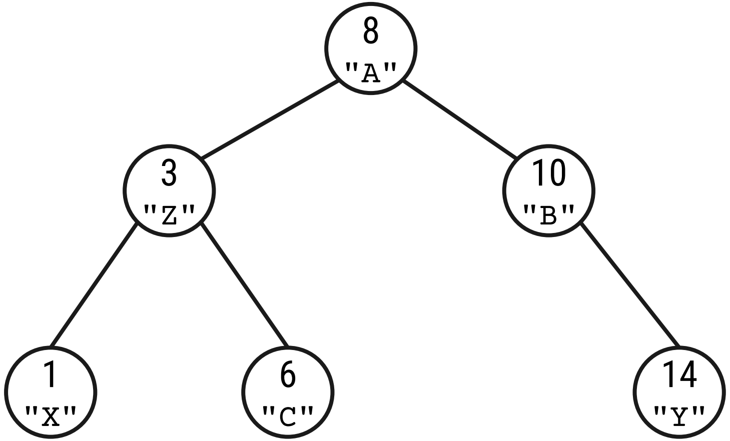
We can utilize this additional property of the keys when implementing a dictionary.

In fact, we can significantly improve upon the performance of a generic map.

Since the keys are ordered, we can implement a dictionary with any type of binary search tree (e.g., splay tree, red-black tree, etc).

- The nodes are (key, value) pairs.
- The nodes are ordered by keys.

This approach allows us to replace the linear run-times with logarithmic run-times.



A binary search tree with (key, value) pairs in each node. The order of the nodes is based on the order of the keys.

get	put	remove	contains Key	contains Value
O(log n)-time	O(log n)-time	O(log n)-time	O(log n)-time	O(n)-time

Worst-case run-times of various dictionary operations. Note that these run-times assume the use of a self-balancing binary search tree with worst-case logarithmic run-times (e.g. red-black and not splay).

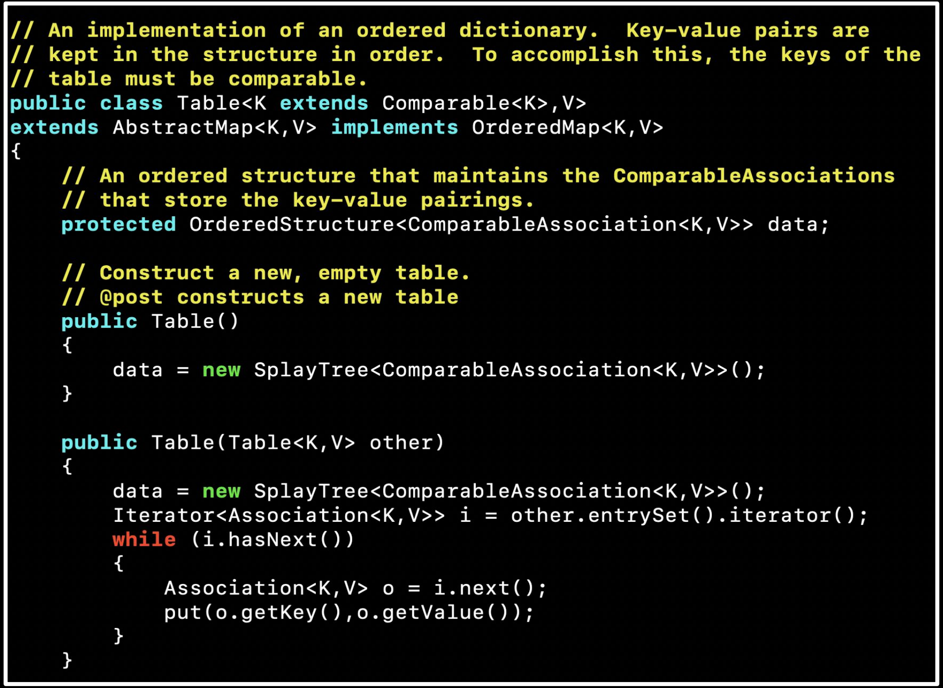
`structure` Package

Implementation of `Table`

In the `structure` package, the term *table* is used instead of dictionary.

Besides using a binary search tree instead of a linked list, the implementation of the `Table` class differs from the implementation of `MapList` in several ways.

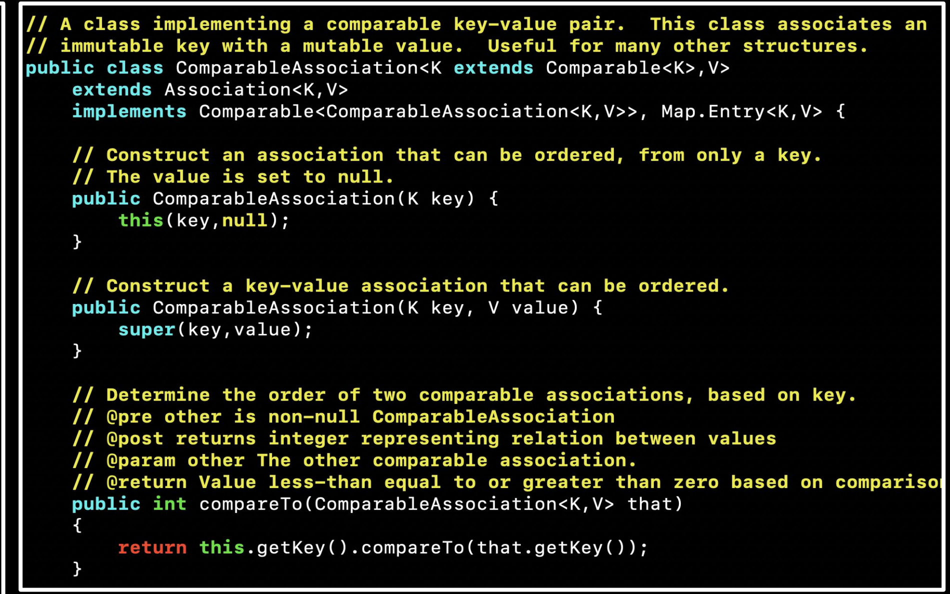
- The interface `OrderedMap` is used instead of `Map`.
- Each (key, value) pair is a `ComparableAssociation` rather than an `Association`.
- `Table` extends `Comparable` and `AbstractMap` whereas `MapList` does not.



```java
// An implementation of an ordered dictionary.  Key-value pairs are
// kept in the structure in order.  To accomplish this, the keys of the
// table must be comparable.
public class Table<K extends Comparable<K>,V>
extends AbstractMap<K,V> implements OrderedMap<K,V>
{
    // An ordered structure that maintains the ComparableAssociations
    // that store the key-value pairings.
    protected OrderedStructure<ComparableAssociation<K,V>> data;

    // Construct a new, empty table.
    // @post constructs a new table
    public Table()
    {
        data = new SplayTree<ComparableAssociation<K,V>>();
    }

    public Table(Table<K,V> other)
    {
        data = new SplayTree<ComparableAssociation<K,V>>();
        Iterator<Association<K,V>> i = other.entrySet().iterator();
        while (i.hasNext())
        {
            Association<K,V> o = i.next();
            put(o.getKey(),o.getValue());
        }
    }
```

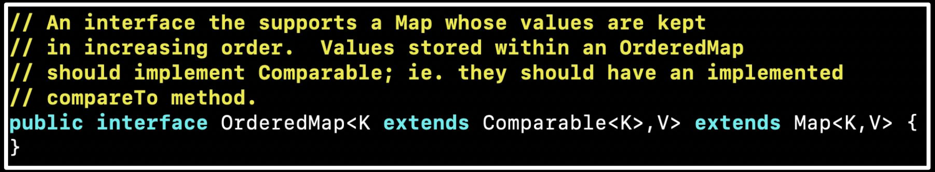


```java
// A class implementing a comparable key-value pair.  This class associates an
// immutable key with a mutable value.  Useful for many other structures.
public class ComparableAssociation<K extends Comparable<K>,V>
    extends Association<K,V>
    implements Comparable<ComparableAssociation<K,V>>, Map.Entry<K,V> {

    // Construct an association that can be ordered, from only a key.
    // The value is set to null.
    public ComparableAssociation(K key) {
        this(key,null);
    }

    // Construct a key-value association that can be ordered.
    public ComparableAssociation(K key, V value) {
        super(key,value);
    }

    // Determine the order of two comparable associations, based on key.
    // @pre other is non-null ComparableAssociation
    // @post returns integer representing relation between values
    // @param other The other comparable association.
    // @return Value less-than equal to or greater than zero based on compariso
    public int compareTo(ComparableAssociation<K,V> that)
    {
        return this.getKey().compareTo(that.getKey());
    }
```



```java
// An interface the supports a Map whose values are kept
// in increasing order.  Values stored within an OrderedMap
// should implement Comparable; ie. they should have an implemented
// compareTo method.
public interface OrderedMap<K extends Comparable<K>,V> extends Map<K,V> {
}
```

The `structure` package's implementation of `Table` (aka, dictionary).