Sets and Maps

Professor Frank Kreimendahl

School of Computing and Data Science
Wentworth Institute of Technology

November 7, 2022

Set ADT
Set Description
Set ADT
Set Review
Set Interface
List Comparisons
Maps

Set ADT

Set Description

- **Set** objects are data collections with specific properties:
  - Sets are not indexed
  - Sets are not ordered
  - Sets implement search and information retrieval efficiently
  - Elements can be removed without shifting other elements
  - Elements are unique

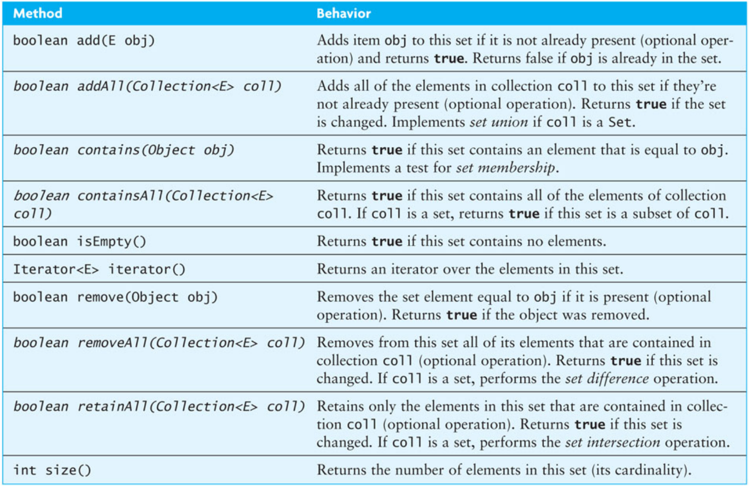
Set ADT

- A **Set** contains no duplicates
- Operations on sets:
  - adding an element
  - removing an element
  - membership query
  - union: $A \cup B$
  - intersection: $A \cap B$
  - difference: $A \setminus B$
  - subset query: $A \subseteq B$

Set Review

- union: $\{1,3,5\} \cup \{1,2,3\} = \{1,2,3,5\}$
- intersection: $\{1,3,5\} \cap \{1,2,3\} = \{1,3\}$
- difference: $\{1,3,5\} \setminus \{1,2,3\} = \{5\}$
- subset query: $\{1,3,5\} \subseteq \{1,2,3\}$ is false, $\{1,3\} \subseteq \{1,2,3\}$ is true

Set Interface



Method	Behavior
boolean add(E obj)	Adds item obj to this set if it is not already present (optional operation) and returns **true**. Returns false if obj is already in the set.
boolean addAll(Collection<E> coll)	Adds all of the elements in collection coll to this set if they're not already present (optional operation). Returns **true** if the set is changed. Implements *set union* if coll is a Set.
boolean contains(Object obj)	Returns **true** if this set contains an element that is equal to obj. Implements a test for *set membership*.
boolean containsAll(Collection<E> coll)	Returns **true** if this set contains all of the elements of collection coll. If coll is a set, returns **true** if this set is a subset of coll.
boolean isEmpty()	Returns **true** if this set contains no elements.
Iterator<E> iterator()	Returns an iterator over the elements in this set.
boolean remove(Object obj)	Removes the set element equal to obj if it is present (optional operation). Returns **true** if the object was removed.
boolean removeAll(Collection<E> coll)	Removes from this set all of its elements that are contained in collection coll (optional operation). Returns **true** if this set is changed. If coll is a set, performs the *set difference* operation.
boolean retainAll(Collection<E> coll)	Retains only the elements in this set that are contained in collection coll (optional operation). Returns **true** if this set is changed. If coll is a set, performs the *set intersection* operation.
int size()	Returns the number of elements in this set (its cardinality).

List Comparisons

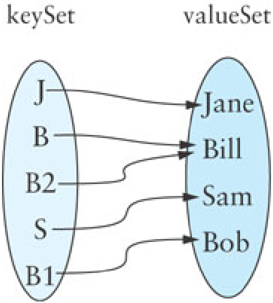
- List elements are not unique so `List.add` operations succeed (with enough memory)
- Set elements may fail on `Set.add` calls if they were already in the set
- Set does not have a `get` operation because elements do not have an index
- Set implements Iterable, but there is no guarantee on the order in which elements are iterated

Maps

Map Description

- **Map**s are related to **Set**s
- A Map is a set of ordered pairs of elements
- These ordered pairs (k, v) have a key first, value second
- Keys must be unique
- Values may have repeats
- The **Map** provides a translation from key to value
- We have seen a special case of a map already: arrays map integers to any type

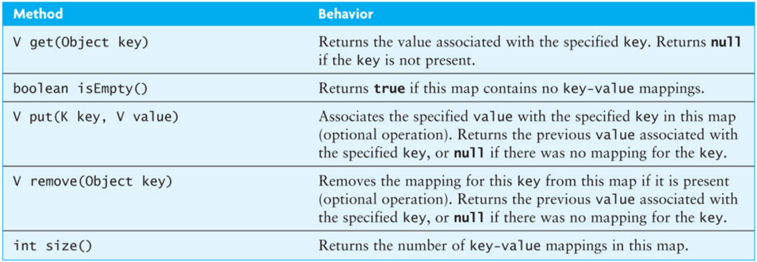
Example Search Path



Map of keys to values

This represents the set of pairs:
$\{(J,Jane),(B,Bill),(S,Sam),(B1,Bob),(B2,Bill)\}$

Map Interface



Method	Behavior
`V get(Object key)`	Returns the value associated with the specified `key`. Returns **`null`** if the `key` is not present.
`boolean isEmpty()`	Returns **`true`** if this map contains no `key-value` mappings.
`V put(K key, V value)`	Associates the specified `value` with the specified `key` in this map (optional operation). Returns the previous `value` associated with the specified `key`, or **`null`** if there was no mapping for the `key`.
`V remove(Object key)`	Removes the mapping for this `key` from this map if it is present (optional operation). Returns the previous `value` associated with the specified `key`, or **`null`** if there was no mapping for the `key`.
`int size()`	Returns the number of `key-value` mappings in this map.

Array Equivalence

- Arrays are maps but use atypical syntax:
  - `myArr[2] = x` translates to `myMap.put(2, x)`
  - `x = myArr[2]` translates to `x = myMap.get(2)`
  - Arrays don't have a built-in `remove` operation
- Maps follow Java's OOP syntax and interface of other Collections
- Sometimes maps are referred to as *associative arrays*

Example Uses For Maps

- Countries: Coordinate → Country Name
- User Database: User ID → User Record
- Academic Papers: doi → article
- Memory: Address → Value
- Note that all of these have `unique` keys. If you have data with repeats of keys (like an English dictionary, first names, age, etc.), a map cannot store their information