Sets & Maps 8A

The `Set` and Map Interfaces

Sets

- A set is a collection of objects that contains no duplicates.
  - The index of each element in the set is not important.
  - The presence of the element is important.
- Set operations:

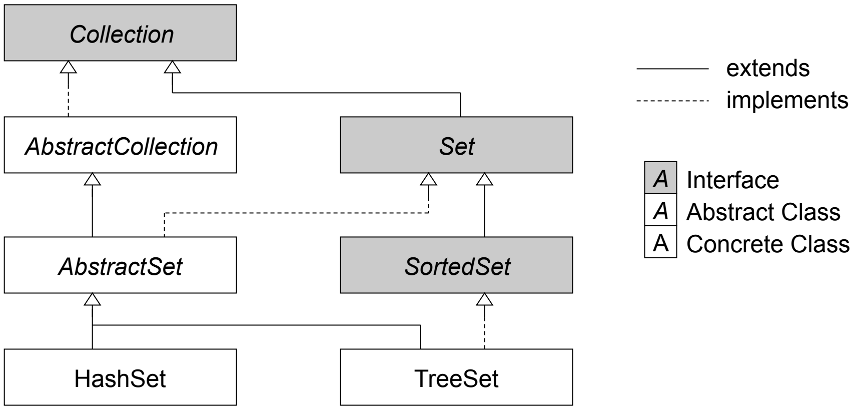
Union	A ∪ B
Intersection	A ∩ B
Difference	A - B
Subset	A ⊂ B

Sets

A = { 1, 2, 4, 8, 16, 32, 64 }
B = { 1, 4, 9, 16, 25, 36, 49, 64 }
(note: elements in the set are not necessarily stored in sorted order)

- $A \cup B$ = { 1, 2, 4, 8, 9, 16, 25, 32, 36, 49, 64 }
- $A \cap B$ = { 1, 4, 16, 64 }
- A - B = { 2, 8, 32 }
- $A \subset B$ = false

Sets in the Java API



Collection

AbstractCollection

Set

AbstractSet

SortedSet

HashSet

TreeSet

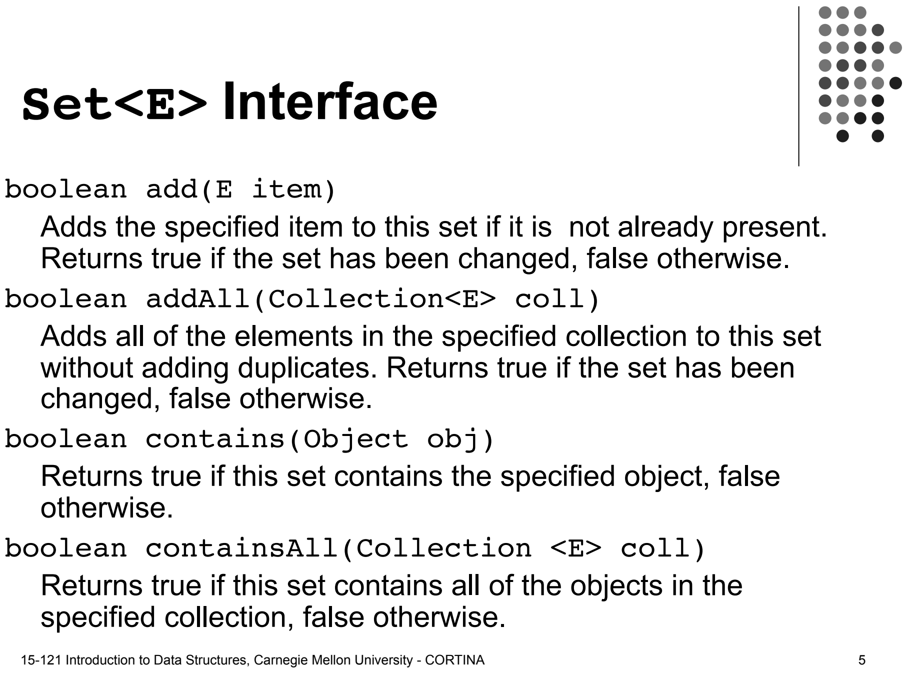
extends

implements

A Interface

A Abstract Class

A Concrete Class



Set<E> Interface

```
boolean add(E item)
```

Adds the specified item to this set if it is not already present. Returns true if the set has been changed, false otherwise.

```
boolean addAll(Collection<E> coll)
```

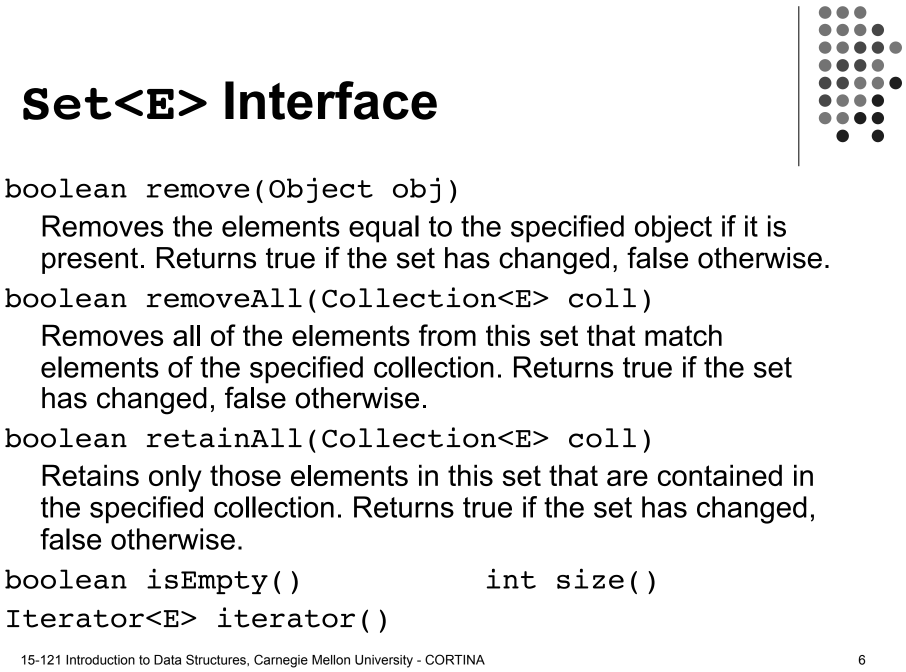
Adds all of the elements in the specified collection to this set without adding duplicates. Returns true if the set has been changed, false otherwise.

```
boolean contains(Object obj)
```

Returns true if this set contains the specified object, false otherwise.

```
boolean containsAll(Collection <E> coll)
```

Returns true if this set contains all of the objects in the specified collection, false otherwise.



Set<E> Interface

```
boolean remove(Object obj)
```

Removes the elements equal to the specified object if it is present. Returns true if the set has changed, false otherwise.

```
boolean removeAll(Collection<E> coll)
```

Removes all of the elements from this set that match elements of the specified collection. Returns true if the set has changed, false otherwise.

```
boolean retainAll(Collection<E> coll)
```

Retains only those elements in this set that are contained in the specified collection. Returns true if the set has changed, false otherwise.

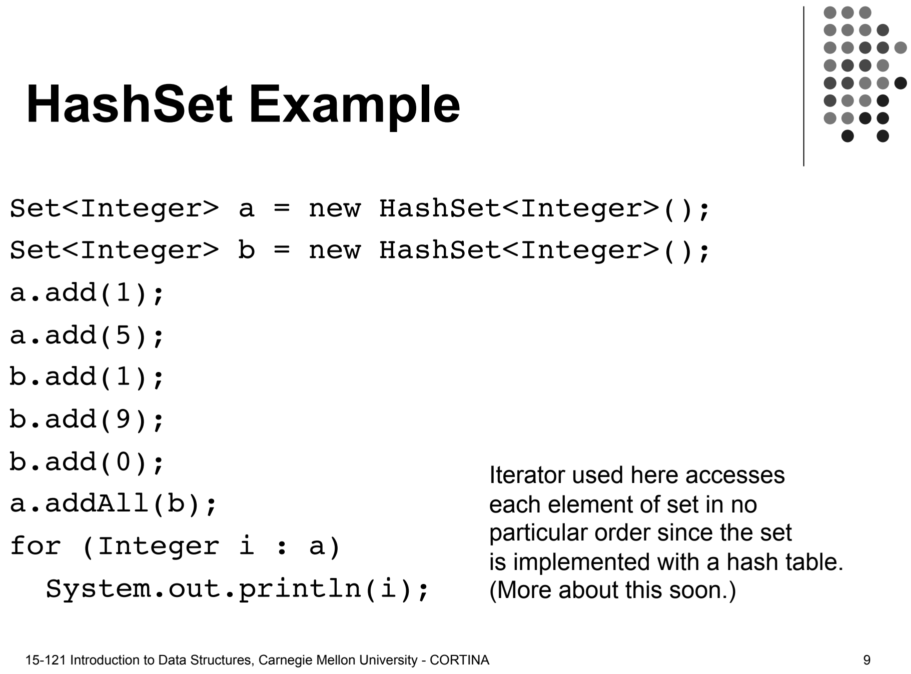
```
boolean isEmpty()            int size()
Iterator<E> iterator()
```

Examples

- Given two sets a and b.
- $a = a \cup b$
- $a = a \cap b$
- $b = b - a$
- if ($b \subset a$)...

HashSet

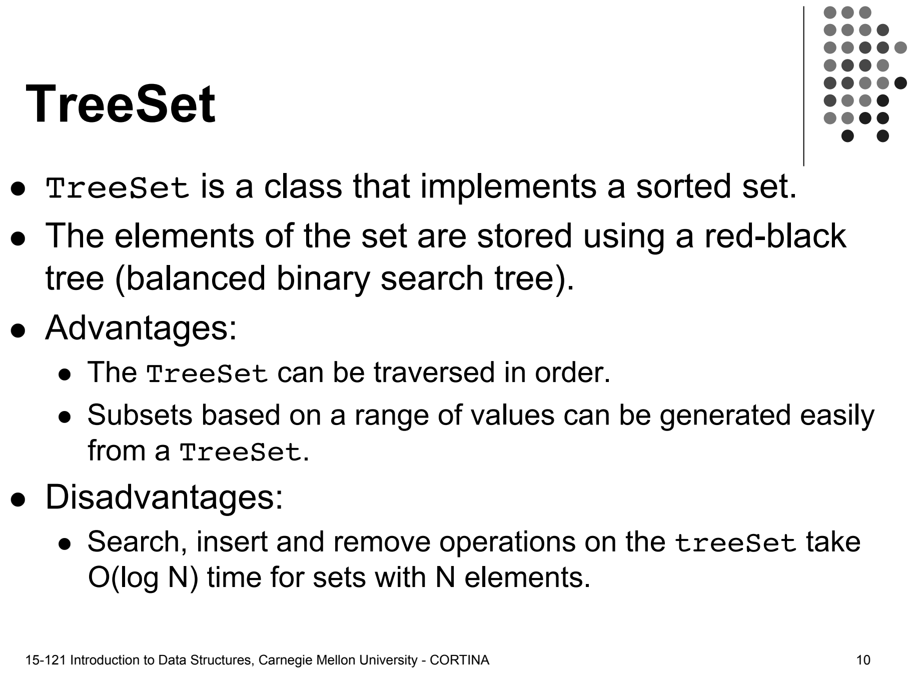
- HashSet is a class that implements a set.
- The elements of the set are stored using a hash table (more about this soon).
- Advantages:
  - The HashSet supports search, insert, and remove operations in O(1) expected time.
- Disadvantages:
  - Traversals cannot be done in a meaningful way with a HashSet.
  - Creating subsets consisting of values within a certain range cannot be done easily.



HashSet Example

```
Set<Integer> a = new HashSet<Integer>();
Set<Integer> b = new HashSet<Integer>();
a.add(1);
a.add(5);
b.add(1);
b.add(9);
b.add(0);
a.addAll(b);
for (Integer i : a)
  System.out.println(i);
```

Iterator used here accesses each element of set in no particular order since the set is implemented with a hash table. (More about this soon.)



TreeSet

- `TreeSet` is a class that implements a sorted set.
- The elements of the set are stored using a red-black tree (balanced binary search tree).
- Advantages:
  - The `TreeSet` can be traversed in order.
  - Subsets based on a range of values can be generated easily from a `TreeSet`.
- Disadvantages:
  - Search, insert and remove operations on the `treeSet` take O(log N) time for sets with N elements.

TreeSet Example

```
SortedSet<Integer> a = new TreeSet<Integer>();
a.add(5);
a.add(3);
a.add(8);
a.add(7);
a.add(1);
for (Integer i : a)
  System.out.println(i);
SortedSet<Integer> b = a.headSet(5);  // b = {1,3}
SortedSet<Integer> c = a.tailSet(5);  // c = {5,7,8}
SortedSet<Integer> d = a.subSet(3,8); // d = {3,5,7}
```

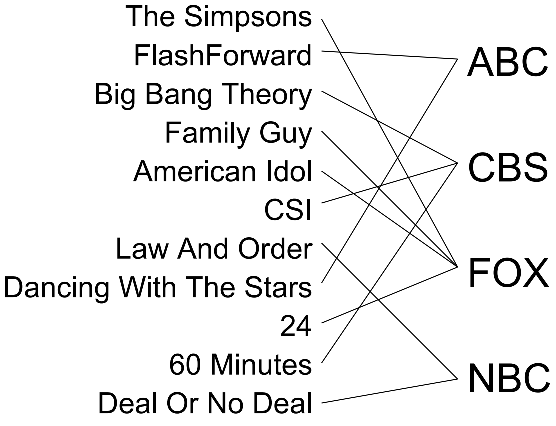
Maps

- A map is a set of ordered pairs.
  - Each ordered pair contains a key and a value.
  - All keys in a map are required to be unique, but all values do not have to be unique.
- Example:

{ ("The Simpsons", "FOX"), ("FlashForward", "ABC"),
("Big Bang Theory", "CBS"), ("Family Guy", "FOX"),
("American Idol", "FOX"), ("CSI", "CBS"),
("Law And Order", "NBC"), ("24", "FOX"),
("Dancing With The Stars", "ABC"),
("60 Minutes", "CBS"), ("Deal Or No Deal", "NBC") }

Maps

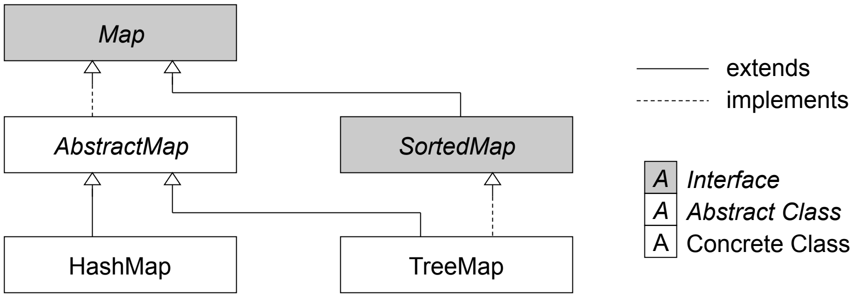
many-to-one mapping
onto mapping



The Simpsons
FlashForward
Big Bang Theory
Family Guy
American Idol
CSI
Law And Order
Dancing With The Stars
24
60 Minutes
Deal Or No Deal

ABC
CBS
FOX
NBC

Maps in the Java API



Map
AbstractMap
SortedMap
HashMap
TreeMap

extends
implements

Interface
Abstract Class
Concrete Class

`Map<K,V>` Interface

`V get(Object key)`

Returns the value associated with the specified key, or null if the key is not present.

`V put(K key, V value)`

Associates the specified value with the specified key in this map. Returns the previous value associated with the specified key, if any, or null.

`V remove(Object key)`

Removes the mapping for the specified key from this map. Returns the previous value associated with the specified key, if any, or null.

`boolean isEmpty()`   `int size()`

HashMap

- `HashMap` is a class that implements a map.
- The (key,value) pairs of the map are stored using a hash table (more about this soon).
- Advantages:
  - The `HashMap` supports search, insert, and remove operations in O(1) expected time.
- Disadvantages:
  - Traversals (using an iterator) cannot be done in a meaningful way with a HashMap.

HashMap Example

```
Map<String, String> tvShowMap
   = new HashMap<String, String>();
tvShowMap.put("The Simpsons", "FOX");
tvShowMap.put("FlashForward", "ABC");
tvShowMap.put("C.S.I.", "CBS");
// C.S.I. changes networks!
tvShowMap.put("C.S.I.", "NBC");
Set<String> showSet = tvShowMap.keySet();
for (String show : showSet)
  System.out.println(show +
   " " + tvShowMap.get(show));
```

```
Output:
C.S.I. NBC
The Simpsons FOX
FlashForward ABC
```

TreeMap

- `TreeMap` is a class that implements a sorted map.
- The (key,value) pairs of the map are stored using a red-black tree (balanced binary search tree).
- Advantages:
  - The `TreeMap` can be traversed in order using an iterator (over a set view of the keys).
  - New maps based on a range of keys can be generated easily from a `TreeMap`.
- Disadvantages:
  - Search, insert and remove operations on the `treeMap` take O(log N) time for sets with N elements.

TreeMap Example

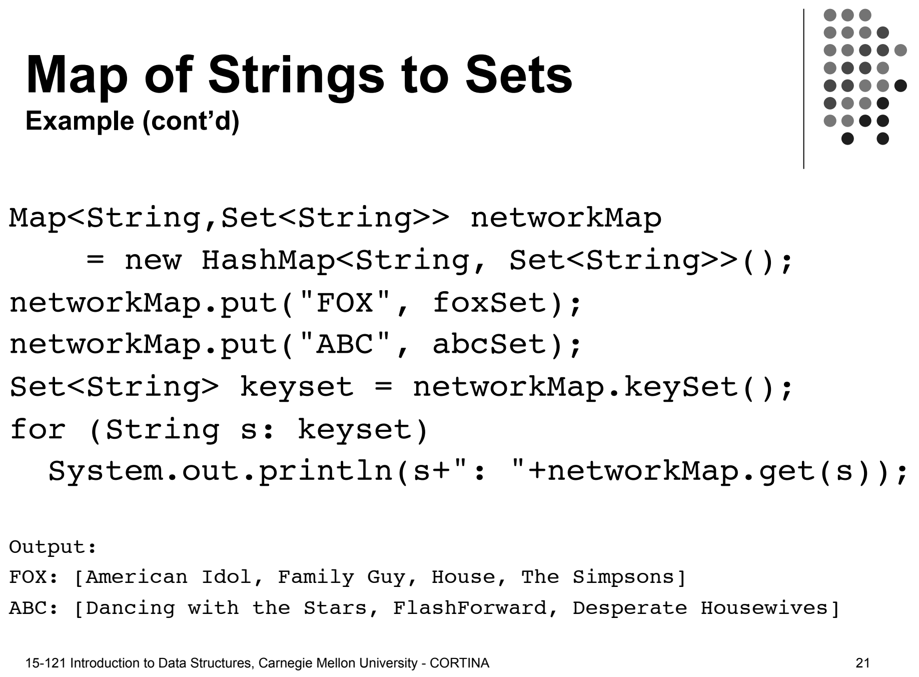
```
Map<String, String> tvShowMap
  = new TreeMap<String,String>();
tvShowMap.put("The Simpsons", "FOX");
tvShowMap.put("FlashForward", "ABC");
tvShowMap.put("C.S.I.", "CBS");
System.out.println("The Simpsons is on " +
  tvShowMap.get("The Simpsons"));
                  Output: The Simpsons is on FOX
Set<String> showSet = tvShowMap.keySet();
for (String show : showSet)
  System.out.println(show +
   "  " + tvShowMap.get(show));
```

```
Output:
C.S.I. CBS
FlashForward ABC
The Simpsons FOX
```

Map of Strings to Sets
Example

```
HashSet<String> abcSet = new HashSet<String>();
abcSet.add("Desperate Housewives");
abcSet.add("FlashForward");
abcSet.add("Dancing with the Stars");

TreeSet<String> foxSet = new TreeSet<String>();
foxSet.add("The Simpsons");
foxSet.add("Family Guy");
foxSet.add("American Idol");
foxSet.add("House");
```



Map of Strings to Sets

Example (cont'd)

```
Map<String,Set<String>> networkMap
    = new HashMap<String, Set<String>>();
networkMap.put("FOX", foxSet);
networkMap.put("ABC", abcSet);
Set<String> keyset = networkMap.keySet();
for (String s: keyset)
  System.out.println(s+": "+networkMap.get(s));
```

```
Output:
FOX: [American Idol, Family Guy, House, The Simpsons]
ABC: [Dancing with the Stars, FlashForward, Desperate Housewives]
```