CSIS 3103

Ch 7: Sets and Maps

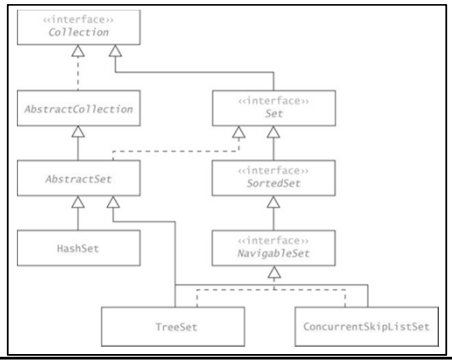
The Set Abstraction

A set is a collection with no duplicate elements

Set operations:

- Test for element membership
- Add elements
- Remove elements
- Union
- Intersection
- Difference
- Subset

Sets and the `Set` Interface



The `Set` Interface and Methods

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).

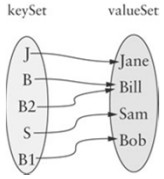
Set Iteration

You can iterate through all elements in a `Set` using an `Iterator` object, but the elements will be accessed in arbitrary order

```
for (String nextItem : setA) {
  // Do something with nextItem
  …
}
```

Maps and the `Map` Interface

- A `Map` is a set of ordered pairs of the form (key, value)
- Keys are required to be unique, but values are not necessarily unique (*many-to-one*)
- Each key is a "mapping" to a particular value
- Maps can enable efficient storage and retrieval of data in a table



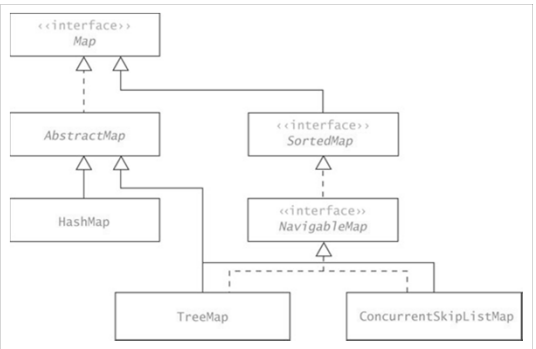
`{(J, Jane),(B, Bill),(S, Sam), (B1, Bob), (B2, Bill)}`

Maps and the Map Interface

Table data often requires a unique ID
This unique ID would equivalent to a key

Type of item	Key	Value
University student	Student ID number	Student name, address, major, grade point average
Online store customer	E-mail address	Customer name, address, credit card information, shopping cart
Inventory item	Part ID	Description, quantity, manufacturer, cost, price

Map Hierarchy

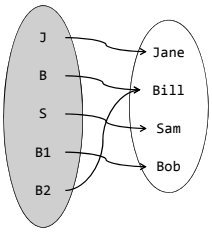


The Map Interface

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.

The Map Interface

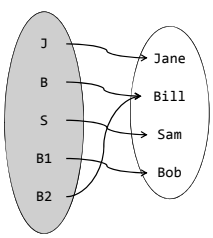
```
Map<String, String> aMap = new
  HashMap<String, String>();

aMap.put("J", "Jane");
aMap.put("B", "Bill");
aMap.put("S", "Sam");
aMap.put("B1", "Bob");
aMap.put("B2", "Bill");
```



The Map Interface

```
aMap.get("B1")
```
returns: `"Bob"`

```
aMap.get("Bill")
```
returns: `null`



Creating an Index of Words

- Section 6.4 used a binary search tree to store an index of words occurring in a term paper
- Each element in the binary search tree consisted of had a word followed by a line number
- Elements in a `Map`, can store all the line numbers for a word in a single index entry