Data Structures

Set and Map

CS284

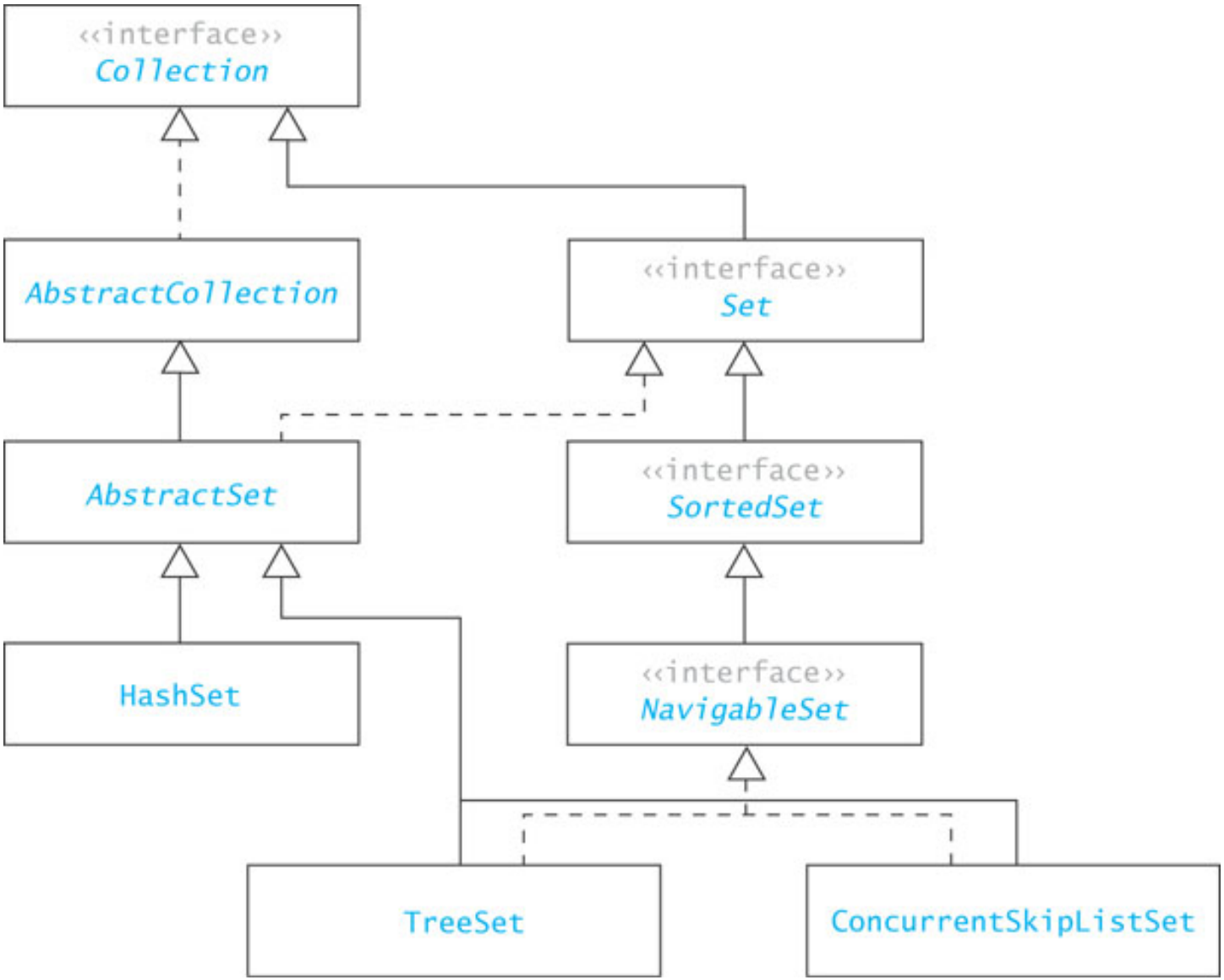
Sets and the `Set<E>` Interface

Maps and the `Map<K,V>` Interface

Introduction

- We now consider another part of the Collection hierarchy
  - the `Set` interface and the classes that implement it
- `Set` objects
  - are not indexed
  - do not reveal the order of insertion of items
  - enable efficient search and retrieval of information
  - allow removal of elements without moving other elements around

The `Set` Interface



The `Set<E>` Interface

- `boolean add(E e)`
- `boolean contains(Object o)`
- `boolean equals(Object o)`

  Compares the specified object with this set for equality.
- `boolean isEmpty()`
- `boolean remove(Object o)`
- `int size()`

The `Set<E>` Interface (cont.)

- `boolean addAll(Collection<? extends E> c)`
- `void clear()`
- `boolean containsAll(Collection<?> c)`
- `int hashCode()`
- `Iterator<E> iterator()`
  Returns an iterator over the elements in this set.
- `boolean removeAll(Collection<?> c)`
- `boolean retainAll(Collection<?> c)`
  Retains only the elements in this set that are contained in the specified collection.
- `Object[] toArray()`

Example 1

- $setA = \{Sally, Ann, Jill\}$ and $setB = \{Jill, Bill, Bob, Ann\}$

```
setA.addAll(setB);
System.out.println(setA);
// Outputs: [Bill, Jill, Ann, Sally, Bob]
```

- If a copy of original setA is in setACopy, then . . .

```
setACopy.retainAll(setB);
System.out.println(setACopy);
// Outputs: [Jill, Ann]
```

```
setACopy.removeAll(setB);
System.out.println(setACopy);
// Outputs: [Sally]
```

Example 2

```
public static void main(String[] args) {
  String[] listA = {"Ann", "Sally", "Jill", "Sally"};
  String[] listB = {"Bob", "Bill", "Ann", "Jill"};
  Set<String> setA = new HashSet<String>();
  Set<String> setAcopy = new HashSet<String>();
  Set<String> setB = new HashSet<String>();

  // Load sets from arrays.
  for (String s:listA) {
      setA.add(s);
      setAcopy.add(s);
  }
  for (String s:listB) { setB.add(s); }
  System.out.println("The 2 sets are: " + "\n" + setA
          + "\n" + setB);

  setA.addAll(setB); // Set union
  setAcopy.retainAll(setB); // Set intersection
  System.out.println("Union: " + setA);
  System.out.println("Intersection: " + setAcopy);
}
```

Example 3: Generating Unique Random Numbers

```
public Set<Integer> pickRandom(int n, int k) {
    Random random = new Random();
    Set<Integer> picked = new HashSet<>();
    while(picked.size() < n) {
        picked.add(random.nextInt(k + 1));
    }
    return picked;
}
```

Src: Stackoverflow

Comparison of Lists and Sets

- Collections implementing the `Set` interface must contain unique elements
- Unlike the `List.add` method, the `Set.add` method returns **`false`** if you attempt to insert a duplicate item
- Unlike a list, `Set` does not have a `get` method – elements cannot be accessed by index
- 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
        ...
}
```

Sets and the `Set<E>` Interface

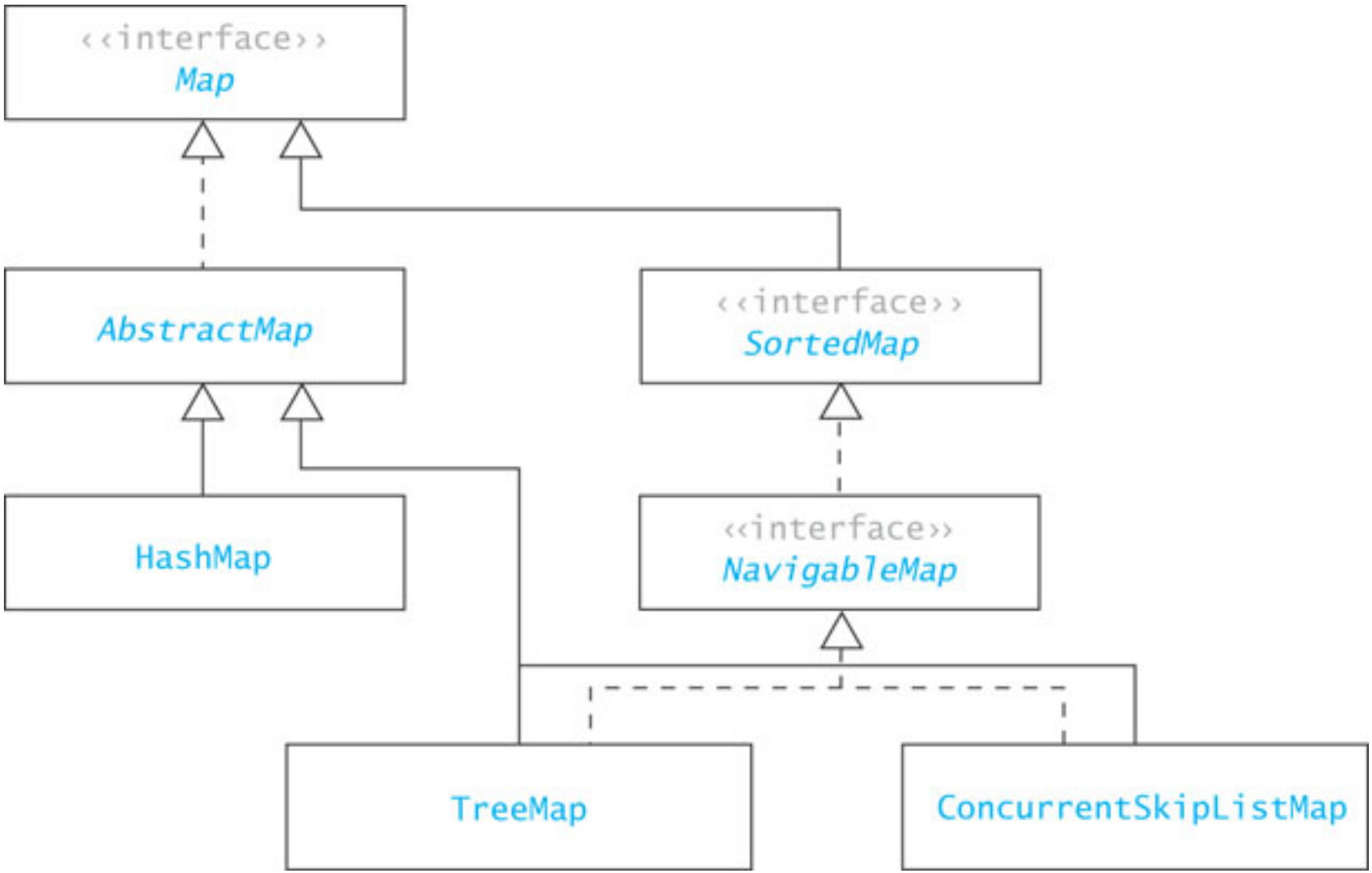
Maps and the `Map<K,V>` Interface

Maps and the `Map` Interface

A `Map`:

- Is a set of ordered pairs whose elements are known as the key and the value
  - Keys must be unique, but values need not be unique
  - You can think of each key as a "mapping" to a particular value
  - valueSet is the set of all the values
  - keySet is the set of all the keys
- Provides efficient storage and retrieval of information in a table
- Can have many-to-one mapping: (B, Bill), (B2, Bill)
  - In an onto mapping, all the elements of valueSet have a corresponding member in keySet

The `Map` Interface



Map Interface `Map<K,V>` (1/2)

- `V get(Object key)`
  Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
- `V put(K key, V value)`
- `boolean isEmpty()`
- `V remove(Object key)`
  Removes the mapping for a key from this map if it is present (optional operation).
- `int size()`

Map Interface `Map<K,V>` (2/2)

- `void clear()`
- `boolean containsKey(Object key)`
- `boolean containsValue(Object value)`
- `Set<Map.Entry<K,V>> entrySet()`
  Returns a Set view of the mappings contained in this map.
- `Set<K> keySet()`
  Returns a Set view of the keys contained in this map.

Note: `Entry<K,V>` is a static inner interface

Example

```
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");

aMap.get("B1");   // returns: "Bob"
aMap.get("Bill"); // returns: null, ("Bill" is a value, not a key)
```

Example

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

for (Map.Entry<String,String> entry : aMap.entrySet()) {
   String key = entry.getKey();
   String value = entry.getValue();
   System.out.println(key+value);
}
```