CS112
Maps
Chapters 21

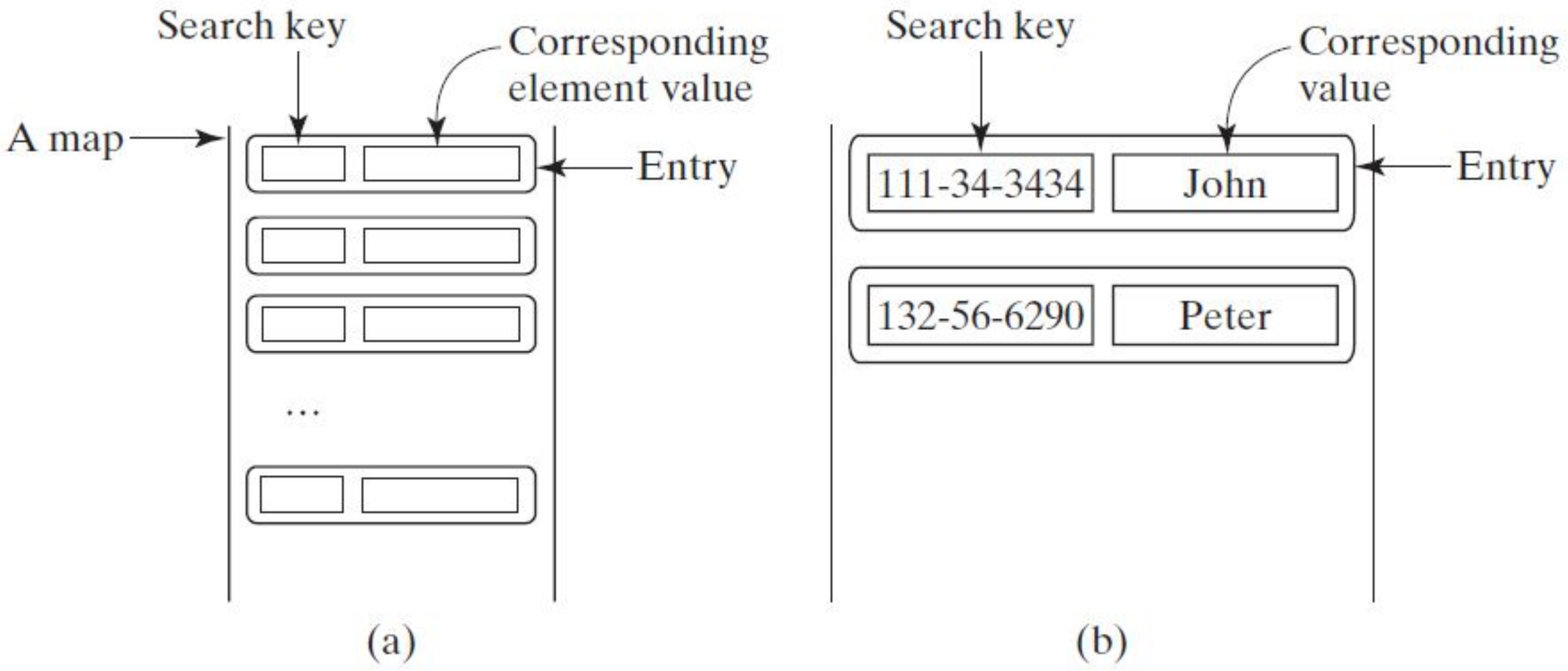
Lecture 19

الفصل الدراسي الثاني 1443- Spring 2022

College of Computer Science and Engineering

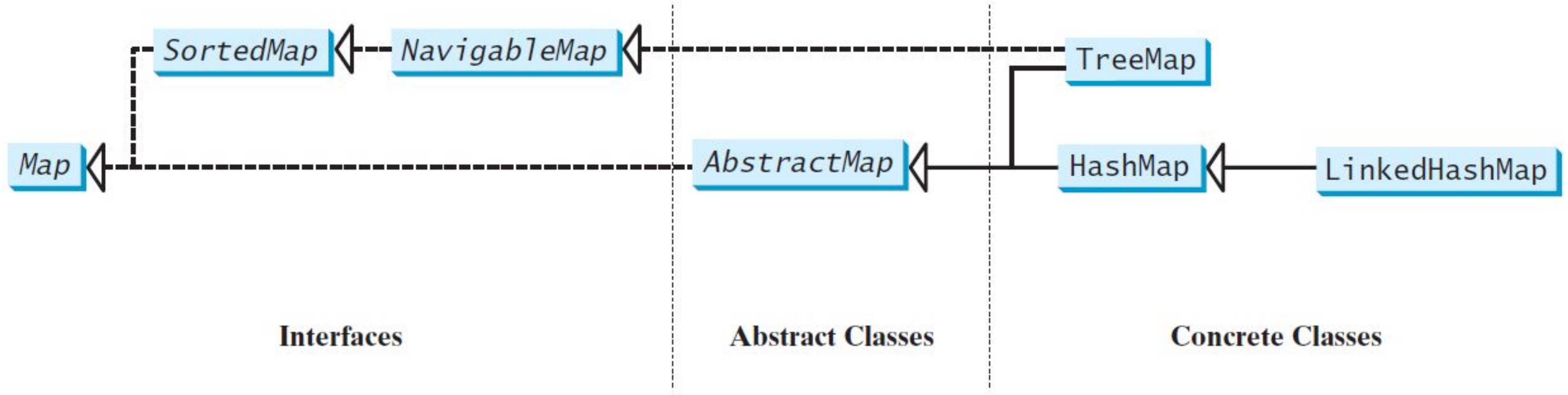
The Map Interface

- The Map interface maps keys to the elements. The keys are like indexes. In List, the indexes are integer. In Map, the keys can be any objects.

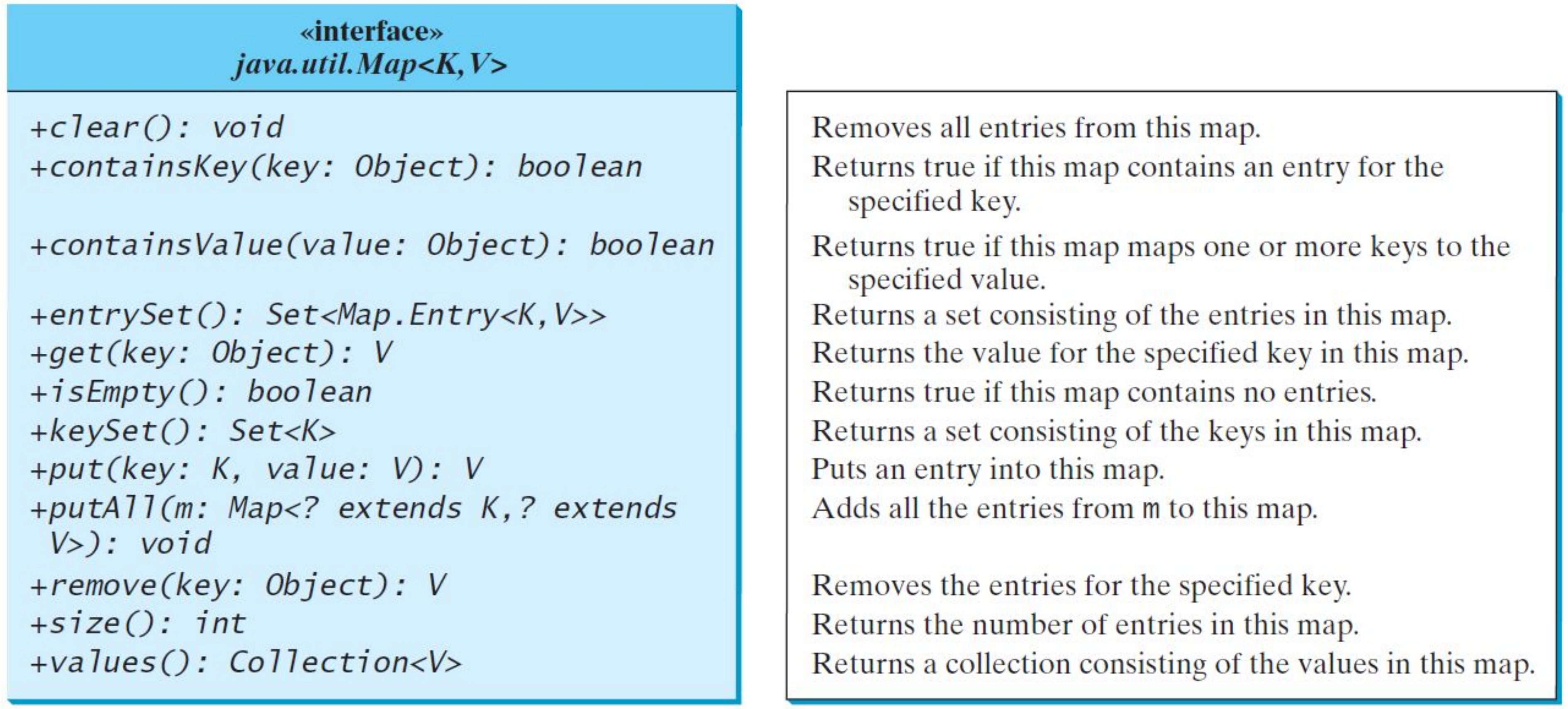


Map Interface and Class Hierarchy

- An instance of Map represents a group of objects, each of which is associated with a key. You can get the object from a map using a key, and you have to use a key to put the object into the map.



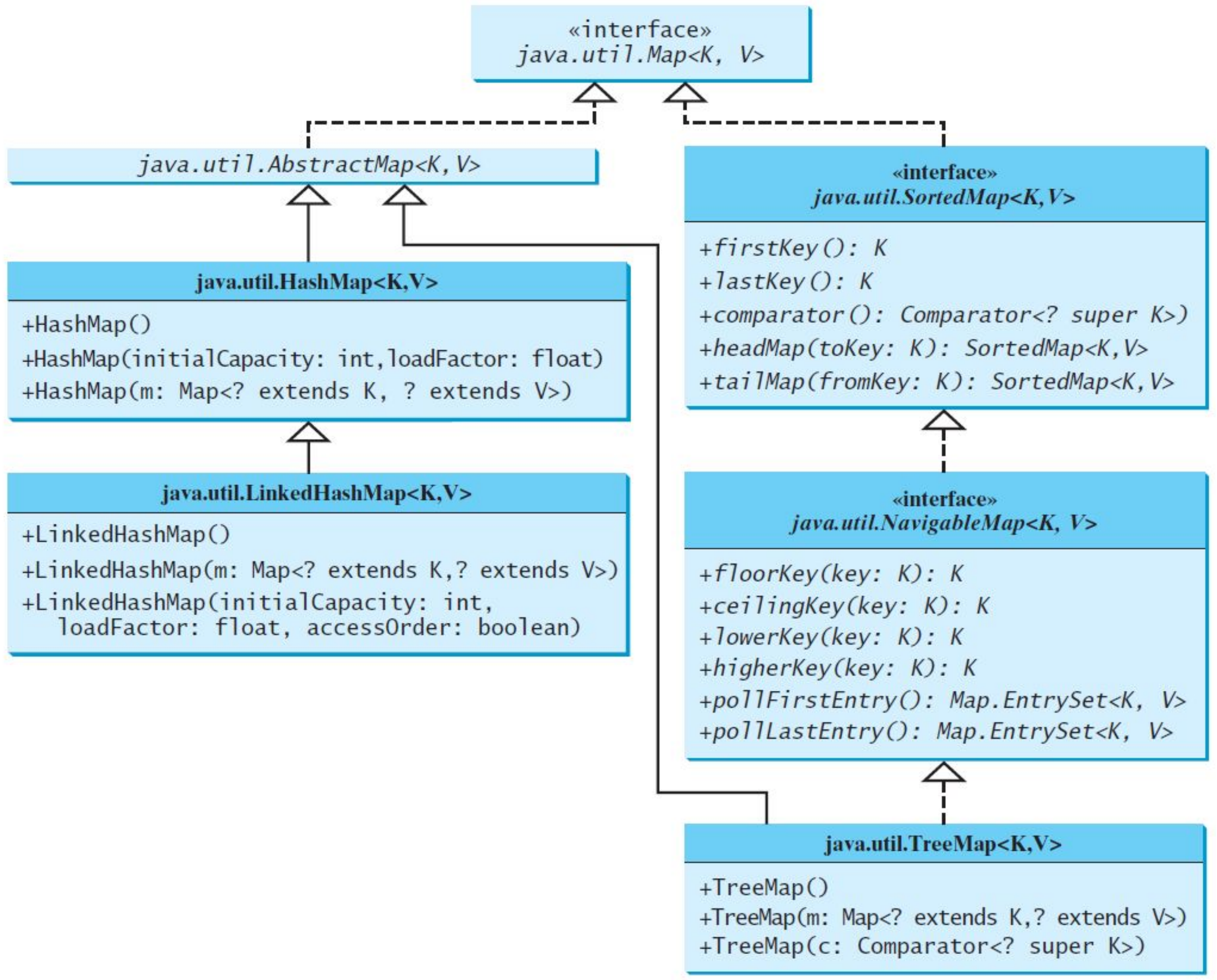
The Map Interface UML Diagram



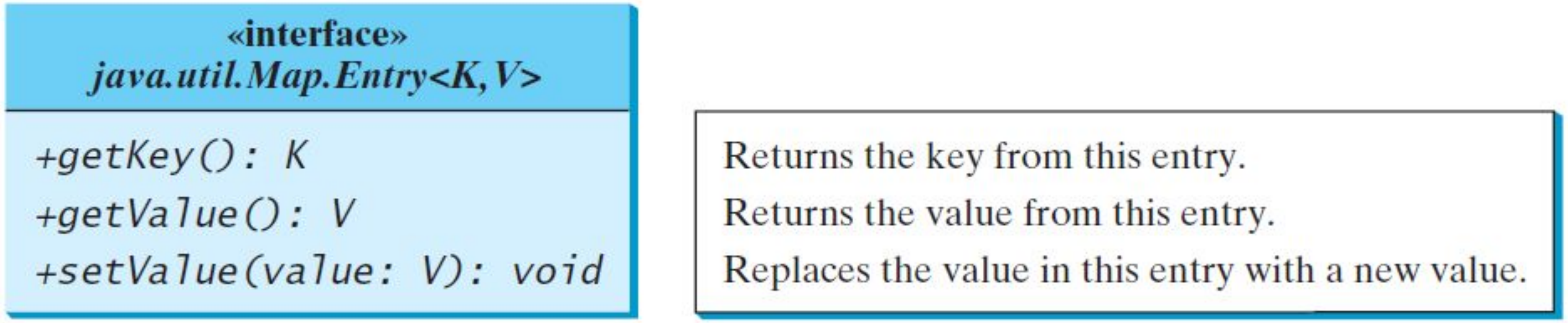
«interface»
java.util.Map<K,V>

`+clear(): void`	Removes all entries from this map.
`+containsKey(key: Object): boolean`	Returns true if this map contains an entry for the specified key.
`+containsValue(value: Object): boolean`	Returns true if this map maps one or more keys to the specified value.
`+entrySet(): Set<Map.Entry<K,V>>`	Returns a set consisting of the entries in this map.
`+get(key: Object): V`	Returns the value for the specified key in this map.
`+isEmpty(): boolean`	Returns true if this map contains no entries.
`+keySet(): Set<K>`	Returns a set consisting of the keys in this map.
`+put(key: K, value: V): V`	Puts an entry into this map.
`+putAll(m: Map<? extends K,? extends V>): void`	Adds all the entries from `m` to this map.
`+remove(key: Object): V`	Removes the entries for the specified key.
`+size(): int`	Returns the number of entries in this map.
`+values(): Collection<V>`	Returns a collection consisting of the values in this map.

Concrete Map Classes


«interface»
java.util.Map<K, V>
java.util.AbstractMap<K,V>
«interface»
java.util.SortedMap<K,V>
+firstKey(): K
+lastKey(): K
+comparator(): Comparator<? super K>)
+headMap(toKey: K): SortedMap<K,V>
+tailMap(fromKey: K): SortedMap<K,V>
java.util.HashMap<K,V>
+HashMap()
+HashMap(initialCapacity: int,loadFactor: float)
+HashMap(m: Map<? extends K, ? extends V>)
java.util.LinkedHashMap<K,V>
+LinkedHashMap()
+LinkedHashMap(m: Map<? extends K,? extends V>)
+LinkedHashMap(initialCapacity: int, loadFactor: float, accessOrder: boolean)
«interface»
java.util.NavigableMap<K, V>
+floorKey(key: K): K
+ceilingKey(key: K): K
+lowerKey(key: K): K
+higherKey(key: K): K
+pollFirstEntry(): Map.EntrySet<K, V>
+pollLastEntry(): Map.EntrySet<K, V>
java.util.TreeMap<K,V>
+TreeMap()
+TreeMap(m: Map<? extends K,? extends V>)
+TreeMap(c: Comparator<? super K>)

Entry


«interface»
java.util.Map.Entry<K,V>

```
+getKey(): K
+getValue(): V
+setValue(value: V): void
```

Returns the key from this entry.
Returns the value from this entry.
Replaces the value in this entry with a new value.

HashMap and TreeMap

- The HashMap and TreeMap classes are two concrete implementations of the Map interface.
- The HashMap class is efficient for locating a value, inserting a mapping, and deleting a mapping.
- The TreeMap class, implementing SortedMap, is efficient for traversing the keys in a sorted order.

LinkedHashMap

- LinkedHashMap was introduced in JDK 1.4. It extends HashMap with a linked list implementation that supports an ordering of the entries in the map.
- The entries in a HashMap are not ordered, but the entries in a LinkedHashMap can be retrieved in the order in which they were inserted into the map (known as the insertion order), or the order in which they were last accessed, from least recently accessed to most recently (access order).
- The no-arg constructor constructs a LinkedHashMap with the insertion order. To construct a LinkedHashMap with the access order, use the LinkedHashMap(initialCapacity, loadFactor, true).

Using HashMap and TreeMap

- This example creates a hash map that maps borrowers to mortgages.
- The program first creates a hash map with the borrower’s name as its key and mortgage as its value.
- The program then creates a tree map from the hash map, and displays the mappings in ascending order of the keys.
  - See TestMap.java

Case Study: Counting the Occurrences of Words in a Text

- This program counts the occurrences of words in a text and displays the words and their occurrences in ascending order of the words.
- The program uses a hash map to store a pair consisting of a word and its count. For each word, check whether it is already a key in the map. If not, add the key and value 1 to the map. Otherwise, increase the value for the word (key) by 1 in the map. To sort the map, convert it to a tree map.
  - See CountOccuranceOfWords.java

Recommended Exercises

- Page 818-820