

CISC 3115

# Map

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

- Discussed
  - Concept of data structure
  - Use data structures
    - List
    - Sorting and searching in lists and arrays
  - Stack
  - Queue and priority queue
  - Set
- To discuss
  - map

# Outline of This Lecture

- Concept of the Map data structure
- Map in Java
- Map, HashMap, LinkedHashMap, and TreeMap
- Examples

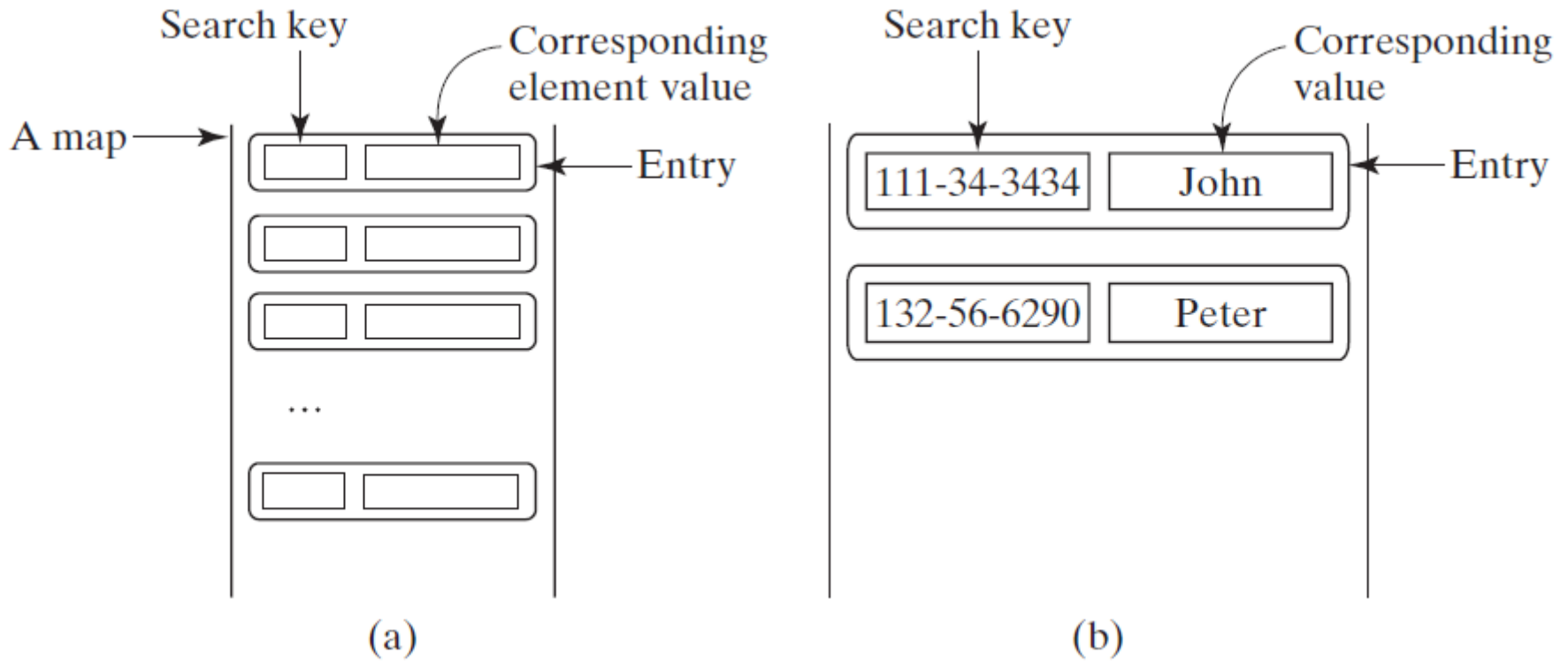
# Motivation

- In many applications, we want to find an element in collection
  - Students enrolled in a class. Find a student given her or his name
  - Passengers on board an airplane. Find a passenger given her or his seat no.
- How may we do it?
  - Sequential search. Inefficient, if we do it a lot
  - Sort and binary search. Great, just remember to sort.
  - Use a map

# The Map Data Structure

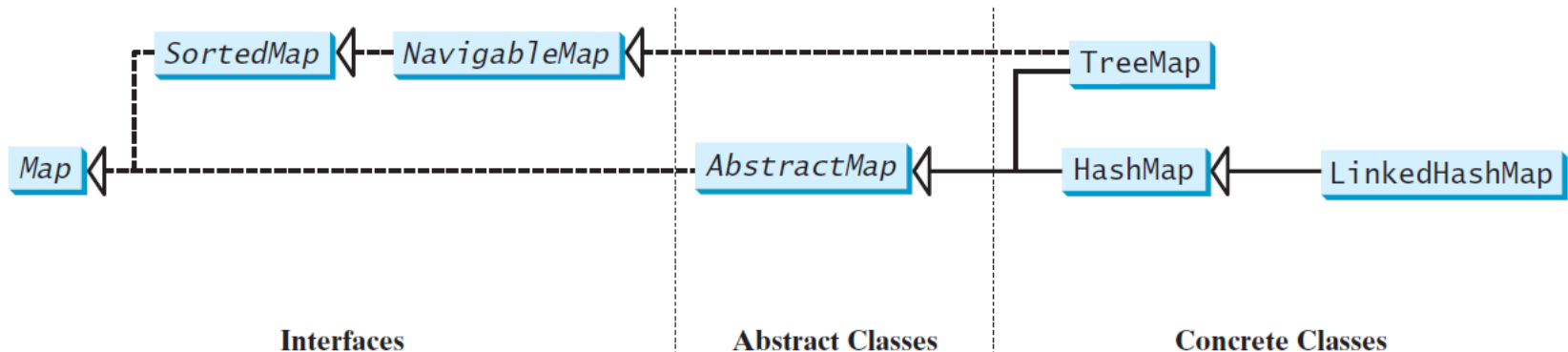
- A map maps keys to values.
- A map cannot contain duplicate keys.
- Each key can map to at most one value.
  - Note: if we remove this constraint, i.e., a key can map to more than one value, we call it a multi-set.
- Compare to List
  - In a list, the indexes are integer. It is like map an integer to the object at the index
  - In a map, it is like that indexes can be any data type. It maps an object to another.
    - Example: a student's name → a Student

# Map and List



# Map Interface and Type Hierarchy

- Map: a group of objects, each of which is associated with a key.
- Must use a key to get the object from a map
- Must use a key to put the object into the map



# Map Interface in Java

«interface»  
*java.util.Map<K, V>*

```
+clear(): void  
+containsKey(key: Object): boolean  
  
+containsValue(value: Object): boolean  
  
+entrySet(): Set<Map.Entry<K, V>>  
+get(key: Object): V  
+isEmpty(): boolean  
+keySet(): Set<K>  
+put(key: K, value: V): V  
+putAll(m: Map<? extends K, ? extends V>): void  
+remove(key: Object): V  
+size(): int  
+values(): Collection<V>
```

Removes all entries from this map.

Returns true if this map contains an entry for the specified key.

Returns true if this map maps one or more keys to the specified value.

Returns a set consisting of the entries in this map.

Returns the value for the specified key in this map.

Returns true if this map contains no entries.

Returns a set consisting of the keys in this map.

Puts an entry into this map.

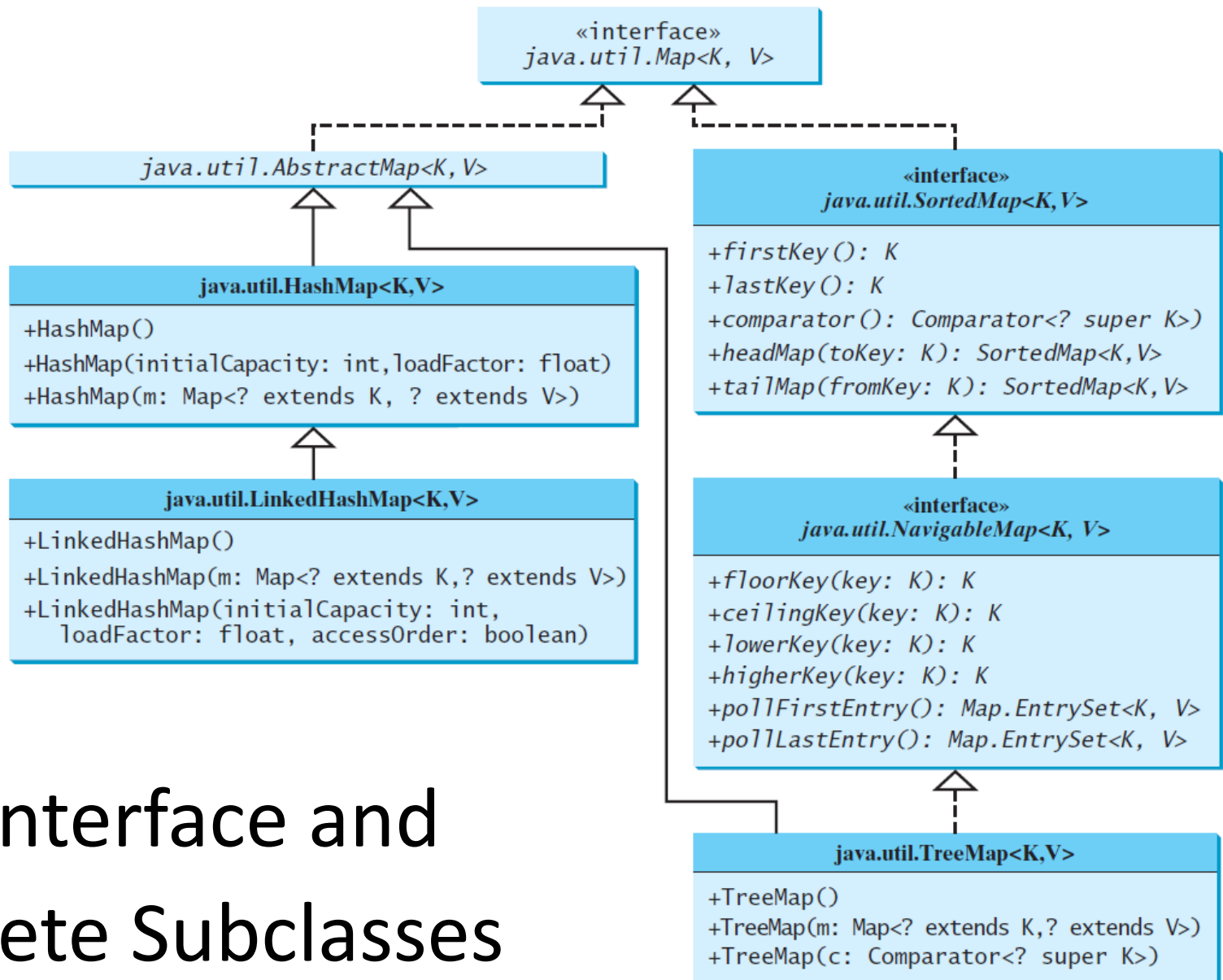
Adds all the entries from *m* to this map.

Removes the entries for the specified key.

Returns the number of entries in this map.

Returns a collection consisting of the values in this map.





# Map Interface and Concrete Subclasses

# Entry Interface

«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

- 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

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

# HashMap and TreeMap: Example 1

- 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

# HashMap and TreeMap: Example 2

- Write an application 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.

# Questions?

- Map, HashMap, LinkedHashMap, TreeMap