Introduction to Java for C++ Programmers

MAP

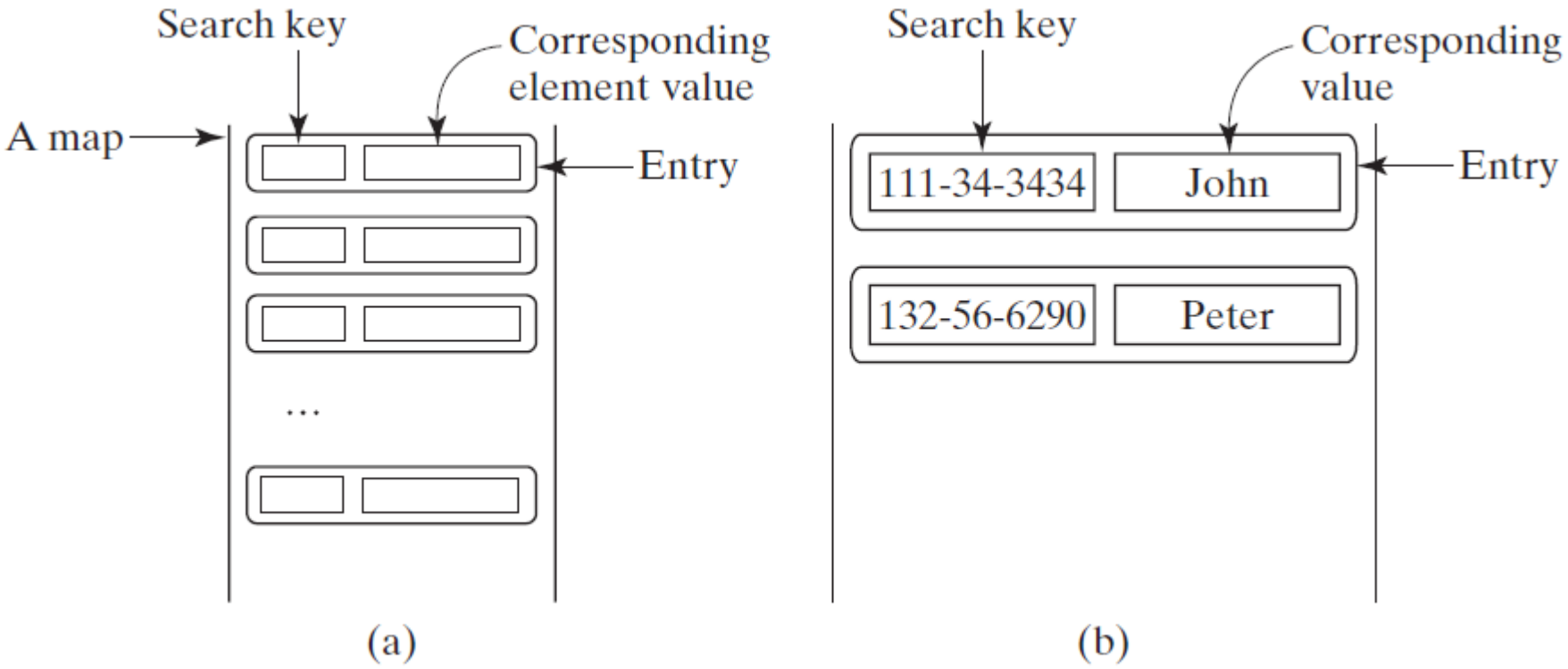
Mahboob Ali

The Map Interface

- Map is a container object that stores a collection of key/ value pairs.
- 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.
- Cannot contain duplicate keys.
- Each key maps to one value.

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

java.util.Map<K, V>	
+clear(): void	Removes all mappings from this map.
+containsKey(key: Object): boolean	Returns true if this map contains a mapping for the specified key.
+containsValue(value: Object): boolean	Returns true if this map maps one or more keys to the specified value.
+entrySet(): Set	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 mappings.
+keySet(): Set<K>	Returns a set consisting of the keys in this map.
+put(key: K, value: V): V	Puts a mapping in this map.
+putAll(m: Map): void	Adds all the mappings from m to this map.
+remove(key: Object): V	Removes the mapping for the specified key.
+size(): int	Returns the number of mappings in this map.
+values(): Collection<V>	Returns a collection consisting of the values in this map.

Implementation

- HashMap:
  - The HashMap class is efficient for locating a value, inserting a mapping, and deleting a mapping.
  - Use it when you want speed and not the order.
  - Entries in the **HashMap** are in random order.
- TreeMap:
  - The TreeMap class, implementing SortedMap, is efficient for traversing the keys in a sorted order.
  - Entries in **TreeMap** are in increasing order of the keys.
- LinkedHashMap:
  - extends HashMap with a linked list implementation, that supports an ordering of the entries in the map.
  - Use it when you need speed nearly as good as HashMap and also give you order.

```
import java.util.*;

public class TestMap {
  public static void main(String[] args) {
    // Create a HashMap
    Map<String, Integer> hashMap = new HashMap<>();
    hashMap.put("Smith", 30);
    hashMap.put("Anderson", 31);
    hashMap.put("Lewis", 29);
    hashMap.put("Cook", 29);

    System.out.println("Display entries in HashMap");
    System.out.println(hashMap + "\n");

    // Create a TreeMap from the preceding HashMap
    Map<String, Integer> treeMap = new TreeMap<>(hashMap);
    System.out.println("Display entries in ascending order of key");
    System.out.println(treeMap);
```

Display entries in HashMap
{Cook=29, Smith=30, Lewis=29, Anderson=31}

Display entries in ascending order of key
{Anderson=31, Cook=29, Lewis=29, Smith=30}

```
  // Create a LinkedHashMap
    Map<String, Integer> linkedHashMap = new LinkedHashMap<>();
    linkedHashMap.put("Smith", 30);
    linkedHashMap.put("Anderson", 31);
    linkedHashMap.put("Lewis", 29);
    linkedHashMap.put("Cook", 29);

    // Display the age for Lewis
    System.out.println("\nThe age for " + "Lewis is " +
      linkedHashMap.get("Lewis"));

    System.out.println("Display entries in LinkedHashMap");
    System.out.println(linkedHashMap);

    // Display each entry with name and age
    System.out.print("\nNames and ages are ");
    treeMap.forEach(
      (name, age) -> System.out.print(name + ": " + age + " "));
  }
}
```

The age for Lewis is 29

Display entries in LinkedHashMap
{Smith=30, Anderson=31, Cook=29, Lewis=29}

Names and ages are Anderson: 31 Cook: 29 Lewis: 29 Smith: 30