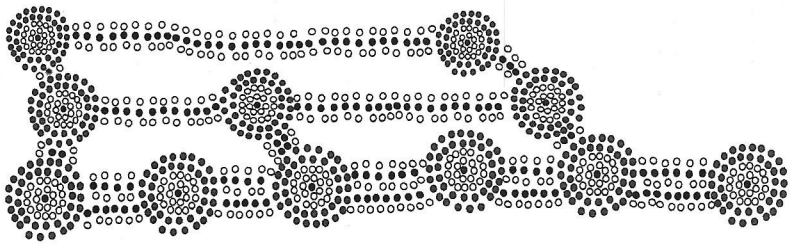
y Queues

eap. Your
d visualize
bottom-up

ory for the
hm is due
loyd [36].
e found in
Reed [70],

Chapter 10 Hash Tables, Maps, and Skip Lists



Contents

10.1 The Map Abstract Data Type

A *map* is an abstract data type designed to efficiently store and retrieve values based upon a uniquely identifying *search key* for each. Specifically, a map stores key-value pairs (k, v), which we call *entries*, where k is the key and v is its corresponding value. Keys are required to be unique, so that the association of keys to values defines a mapping. Figure 10.1 provides a conceptual illustration of a map using the file-cabinet metaphor. For a more modern metaphor, think about the web as being a map whose entries are the web pages. The key of a page is its URL (e.g., http://datastructures.net/) and its value is the page content.

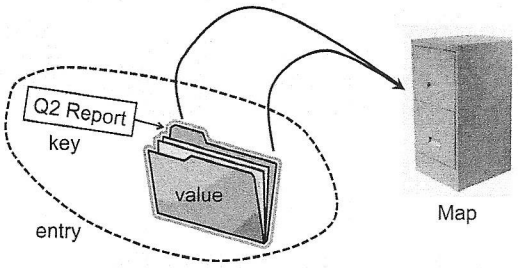


Figure 10.1: A conceptual illustration of the map ADT. Keys (labels) are assigned to values (folders) by a user. The resulting entries (labeled folders) are inserted into the map (file cabinet). The keys can be used later to retrieve or remove values.

Maps are also known as *associative arrays*, because the entry's key serves somewhat like an index into the map, in that it assists the map in efficiently locating the associated entry. However, unlike a standard array, a key of a map need not be numeric, and is does not directly designate a position within the structure. Common applications of maps include the following:

- A university's information system relies on some form of a student ID as a key that is mapped to that student's associated record (such as the student's name, address, and course grades) serving as the value.
- The domain-name system (DNS) maps a host name, such as `www.wiley.com`, to an Internet-Protocol (IP) address, such as `208.215.179.146`.
- A social media site typically relies on a (nonnumeric) username as a key that can be efficiently mapped to a particular user's associated information.
- A company's customer base may be stored as a map, with a customer's account number or unique user ID as a key, and a record with the customer's information as a value. The map would allow a service representative to quickly access a customer's record, given the key.
- A computer graphics system may map a color name, such as `'turquoise'`, to the triple of numbers that describes the color's RGB (red-green-blue) representation, such as (64, 224, 208).

10.1.1 The Map ADT

Since a map stores a collection of objects, it should be viewed as a collection of key-value pairs. As an ADT, a ***map*** M supports the following methods:

size( ): Returns the number of entries in M.

isEmpty( ): Returns a boolean indicating whether M is empty.

get(k): Returns the value v associated with key k, if such an entry exists; otherwise returns null.

put(k, v): If M does not have an entry with key equal to k, then adds entry (k,v) to M and returns null; else, replaces with v the existing value of the entry with key equal to k and returns the old value.

remove(k): Removes from M the entry with key equal to k, and returns its value; if M has no such entry, then returns null.

keySet( ): Returns an iterable collection containing all the keys stored in M.

values( ): Returns an iterable collection containing all the *values* of entries stored in M (with repetition if multiple keys map to the same value).

entrySet( ): Returns an iterable collection containing all the key-value entries in M.

Maps in the java.util Package

Our definition of the map ADT is a simplified version of the java.util.Map interface. For the elements of the iteration returned by entrySet, we will rely on the composite Entry interface introduced in Section 9.2.1 (the java.util.Map relies on the nested java.util.Map.Entry interface).

Notice that each of the operations get(k), put(k, v), and remove(k) returns the existing value associated with key k, if the map has such an entry, and otherwise returns null. This introduces ambiguity in an application for which null is allowed as a natural value associated with a key k. That is, if an entry (k, null) exists in a map, then the operation get(k) will return null, not because it couldn't find the key, but because it found the key and is returning its associated value.

Some implementations of the java.util.Map interface explicitly forbid use of a null value (and null keys, for that matter). However, to resolve the ambiguity when null is allowable, the interface contains a boolean method, containsKey(k) to definitively check whether k exists as a key. (We leave implementation of such a method as an exercise.)

Example 10.1: *In the following, we show the effect of a series of operations on an initially empty map storing entries with integer keys and single-character values.*

Method	*Return Value*	*Map*
isEmpty()	true	$\{\}$
put(5,A)	null	$\{(5,A)\}$
put(7,B)	null	$\{(5,A),(7,B)\}$
put(2,C)	null	$\{(5,A),(7,B),(2,C)\}$
put(8,D)	null	$\{(5,A),(7,B),(2,C),(8,D)\}$
put(2,E)	C	$\{(5,A),(7,B),(2,E),(8,D)\}$
get(7)	B	$\{(5,A),(7,B),(2,E),(8,D)\}$
get(4)	null	$\{(5,A),(7,B),(2,E),(8,D)\}$
get(2)	E	$\{(5,A),(7,B),(2,E),(8,D)\}$
size()	4	$\{(5,A),(7,B),(2,E),(8,D)\}$
remove(5)	A	$\{(7,B),(2,E),(8,D)\}$
remove(2)	E	$\{(7,B),(8,D)\}$
get(2)	null	$\{(7,B),(8,D)\}$
remove(2)	null	$\{(7,B),(8,D)\}$
isEmpty()	false	$\{(7,B),(8,D)\}$
entrySet()	$\{(7,B),(8,D)\}$	$\{(7,B),(8,D)\}$
keySet()	$\{7,8\}$	$\{(7,B),(8,D)\}$
values()	$\{B,D\}$	$\{(7,B),(8,D)\}$

A Java Interface for the Map ADT

A formal definition of a Java interface for our version of the map ADT is given in Code Fragment 10.1. It uses the generics framework (Section 2.5.2), with K designating the key type and V designating the value type.

```
1  public interface Map<K,V> {
2    int size();
3    boolean isEmpty();
4    V get(K key);
5    V put(K key, V value);
6    V remove(K key);
7    Iterable<K> keySet();
8    Iterable<V> values();
9    Iterable<Entry<K,V>> entrySet();
10 }
```

Code Fragment 10.1: Java interface for our simplified version of the map ADT.

10.1.2 Application: Counting Word Frequencies

As a case study for using a map, consider the problem of counting the number of occurrences of words in a document. This is a standard task when performing a statistical analysis of a document, for example, when categorizing an email or news article. A map is an ideal data structure to use here, for we can use words as keys and word counts as values. We show such an application in Code Fragment 10.2.

We begin with an empty map, mapping words to their integer frequencies. (We rely on the ChainHashMap class that will be introduced in Section 10.2.4.) We first scan through the input, considering adjacent alphabetic characters to be words, which we then convert to lowercase. For each word found, we attempt to retrieve its current frequency from the map using the get method, with a yet unseen word having frequency zero. We then (re)set its frequency to be one more to reflect the current occurrence of the word. After processing the entire input, we loop through the entrySet( ) of the map to determine which word has the most occurrences.

```
1  /** A program that counts words in a document, printing the most frequent. */
2  public class WordCount {
3    public static void main(String[ ] args) {
4      Map<String,Integer> freq = new ChainHashMap<>( );  // or any concrete map
5      // scan input for words, using all nonletters as delimiters
6      Scanner doc = new Scanner(System.in).useDelimiter("[^a-zA-Z]+");
7      while (doc.hasNext( )) {
8        String word = doc.next( ).toLowerCase( );    // convert next word to lowercase
9        Integer count = freq.get(word);              // get the previous count for this word
10       if (count == null)
11         count = 0;                                 // if not in map, previous count is zero
12       freq.put(word, 1 + count);                   // (re)assign new count for this word
13     }
14     int maxCount = 0;
15     String maxWord = "no word";
16     for (Entry<String,Integer> ent : freq.entrySet( ))        // find max-count word
17       if (ent.getValue( ) > maxCount) {
18         maxWord = ent.getKey( );
19         maxCount = ent.getValue( );
20       }
21     System.out.print("The most frequent word is '" + maxWord);
22     System.out.println("' with " + maxCount + " occurrences.");
23   }
24 }
```

Code Fragment 10.2: A program for counting word frequencies in a document, printing the most frequent word. The document is parsed using the Scanner class, for which we change the delimiter for separating tokens from whitespace to any nonletter. We also convert words to lowercase.

10.1.3 An AbstractMap Base Class

In the remainder of this chapter (and the next), we will be providing many different implementations of the map ADT using a variety of data structures, each with its own trade-off of advantages and disadvantages. As we have done in earlier chapters, we rely on a combination of abstract and concrete classes in the interest of greater code reuse. Figure 10.2 provides a preview of those classes.

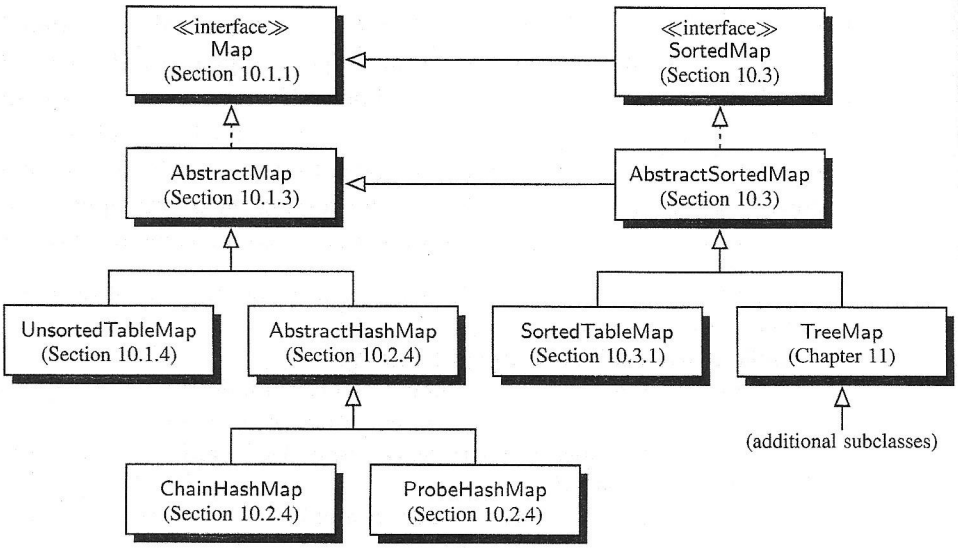


Figure 10.2: Our hierarchy of map types (with references to where they are defined).

We begin, in this section, by designing an AbstractMap base class that provides functionality that is shared by all of our map implementations. More specifically, the base class (given in Code Fragment 10.3) provides the following support:

- An implementation of the isEmpty method, based upon the presumed implementation of the size method.
- A nested MapEntry class that implements the public Entry interface, while providing a composite for storing key-value entries in a map data structure.
- Concrete implementations of the keySet and values methods, based upon an adaption to the entrySet method. In this way, concrete map classes need only implement the entrySet method to provide all three forms of iteration.

  We implement the iterations using the technique introduced in Section 7.4.2 (at that time providing an iteration of all elements of a positional list given an iteration of all positions of the list).

```
1  public abstract class AbstractMap<K,V> implements Map<K,V> {
2    public boolean isEmpty( ) { return size( ) == 0; }
3    //---------- nested MapEntry class ----------
4    protected static class MapEntry<K,V> implements Entry<K,V> {
5      private K k;   // key
6      private V v;   // value
7      public MapEntry(K key, V value) {
8        k = key;
9        v = value;
10     }
11     // public methods of the Entry interface
12     public K getKey( ) { return k; }
13     public V getValue( ) { return v; }
14     // utilities not exposed as part of the Entry interface
15     protected void setKey(K key) { k = key; }
16     protected V setValue(V value) {
17       V old = v;
18       v = value;
19       return old;
20     }
21   } //---------- end of nested MapEntry class ----------
22
23   // Support for public keySet method...
24   private class KeyIterator implements Iterator<K> {
25     private Iterator<Entry<K,V>> entries = entrySet( ).iterator( ); // reuse entrySet
26     public boolean hasNext( ) { return entries.hasNext( ); }
27     public K next( ) { return entries.next( ).getKey( ); }          // return key!
28     public void remove( ) { throw new UnsupportedOperationException( ); }
29   }
30   private class KeyIterable implements Iterable<K> {
31     public Iterator<K> iterator( ) { return new KeyIterator( ); }
32   }
33   public Iterable<K> keySet( ) { return new KeyIterable( ); }
34
35   // Support for public values method...
36   private class ValueIterator implements Iterator<V> {
37     private Iterator<Entry<K,V>> entries = entrySet( ).iterator( ); // reuse entrySet
38     public boolean hasNext( ) { return entries.hasNext( ); }
39     public V next( ) { return entries.next( ).getValue( ); }        // return value!
40     public void remove( ) { throw new UnsupportedOperationException( ); }
41   }
42   private class ValueIterable implements Iterable<V> {
43     public Iterator<V> iterator( ) { return new ValueIterator( ); }
44   }
45   public Iterable<V> values( ) { return new ValueIterable( ); }
46 }
```

Code Fragment 10.3: Implementation of the AbstractMap base class.