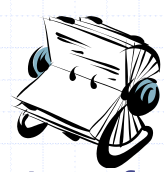
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Maps

Maps



- A map models a searchable collection of key-value entries
- The main operations of a map are for searching, inserting, and deleting items
- Multiple entries with the same key are not allowed
- Applications:
  - address book
  - student-record database

The Map ADT

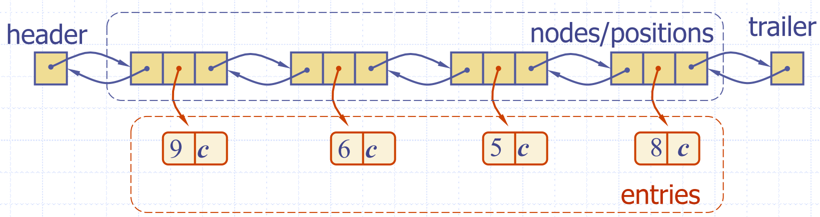
- get(k): if the map M has an entry with key k, return its associated value; else, return null
- put(k, v): insert entry (k, v) into the map M; if key k is not already in M, then return null; else, return old value associated with k
- remove(k): if the map M has an entry with key k, remove it from M and return its associated value; else, return null
- size(), isEmpty()
- entrySet(): return an iterable collection of the entries in M
- keySet(): return an iterable collection of the keys in M
- values(): return an iterator of the values in M

Example

Operation	***Output***	***Map***
isEmpty()	**true**	Ø
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*)
isEmpty()	**false**	(7,*B*),(8,*D*)

A Simple List-Based Map

- We can implement a map using an unsorted list
  - We store the items of the map in a list S (based on a doublylinked list), in arbitrary order



The get(k) Algorithm

Algorithm get(k):
    B = S.positions() {B is an iterator of the positions in S}
    while B.hasNext() **do**
        p = B.next() { the next position in B }
        if p.element().getKey() = k **then**
            return p.element().getValue()
    return null {there is no entry with key equal to k}

The put(k,v) Algorithm

```
Algorithm put(k,v):
B  = S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k  then
        t = p.element().getValue()
        S.set(p,(k,v))
        return t         {return the old value}
S.addLast((k,v))
n = n + 1        {increment variable storing number of entries}
return null      { there was no entry with key equal to k }
```


The remove(k) Algorithm

```
Algorithm remove(k):
B =S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k  then
        t = p.element().getValue()
        S.remove(p)
        n = n – 1     {decrement number of entries}
        return t      {return the removed value}
return null           {there is no entry with key equal to k}
```


Performance of a List-Based Map

- Performance:
  - put takes $O(1)$ time since we can insert the new item at the beginning or at the end of the sequence
  - get and remove take $O(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key
- The unsorted list implementation is effective only for maps of small size or for maps in which puts are the most common operations, while searches and removals are rarely performed (e.g., historical record of logins to a workstation)

