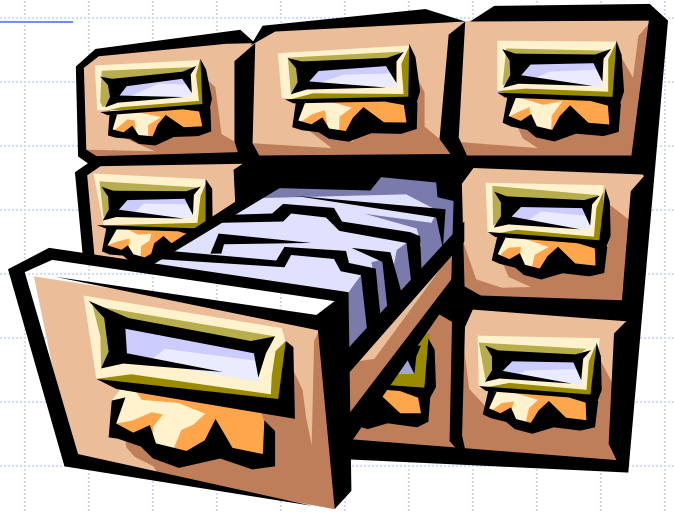
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Maps



Reading

M. T. Goodrich, R. Tamassia and M. H. Goldwasser, *Data Structures and Algorithms in Java*, 6th Edition, 2014.

- **Chapter 10. Hash Tables, Maps and Skip Lists**
- **Sections 10.1 and 10.2**
- **pp. 369-395**

Learning Objectives

- To be able to understand and describe the Map ADT;
- To be able to analyze the complexity of the Map ADT methods;
- To be able to implement the Map ADT with a hash table;
- To be able to understand and apply collision handling methods;
- To be able to apply the Map ADT and hashing methods.

Map 是一种抽象数据类型，用于**根据唯一的搜索键**（key）来**存储和检索值**（value）。
每个元素是一个**键值对**(k, v)，称为 **entry**。

Maps

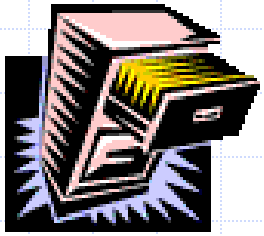
键（key）必须唯一：不允许出现两个相同的键。
值（value）可以重复，但每个键最多对应一个值。
映射（mapping）：key 映射到 value 的关联关系。

- A map is an abstract data type designed to efficiently store and retrieve values based on a ***uniquely identifying search key*** for each.
- A map stores key-value pairs (k, v), which are called ***entries***.
- Keys are required to be ***unique***. The association of keys to values defines a ***mapping***.
- Multiple entries with the same key are not allowed.
- Applications:
  - address book
  - student-record database (every student ID is a key)

通讯录（address book）：name phone number
学生记录系统（student-record database）：student ID student data

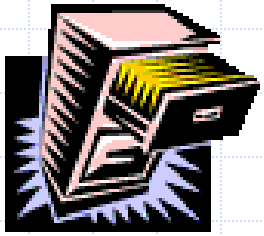
1. get(k)：根据键 k 查找并返回对应的值；若不存在，则返回 null。
2. put(k, v)：插入或更新键值对；若 k 已存在，则更新并返回旧值；否则添加并返回 null。
3. remove(k)：删除键为 k 的条目；若存在则返回对应的值，若不存在则返回 null。

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): if key k is not already in the map M, then *add* entry (k, v) into M and *return* null; else *replace* the old entry with the same key with the new entry and *return* the 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

The Map ADT continued



- 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
- size(), isEmpty()

集合访问方法：
entrySet()：返回所有键值对（entries）的可迭代集合。
keySet()：返回所有键的可迭代集合。
values()：返回所有值的迭代器。
状态检查方法：
size()：返回 Map 中的条目数量。
isEmpty()：判断 Map 是否为空。
用途：这些方法用于遍历、检查或批量处理 Map 中的内容，而不仅限于单个元素操作。

Example

Operation	*Output*	*Map*
isEmpty()		
put(5,*A*)		
put(7,*B*)		
put(2,*C*)		
put(8,*D*)		
put(2,*E*)		
get(7)		
get(4)		
get(2)		
size()		
remove(5)		
remove(2)		
get(2)		
isEmpty()		

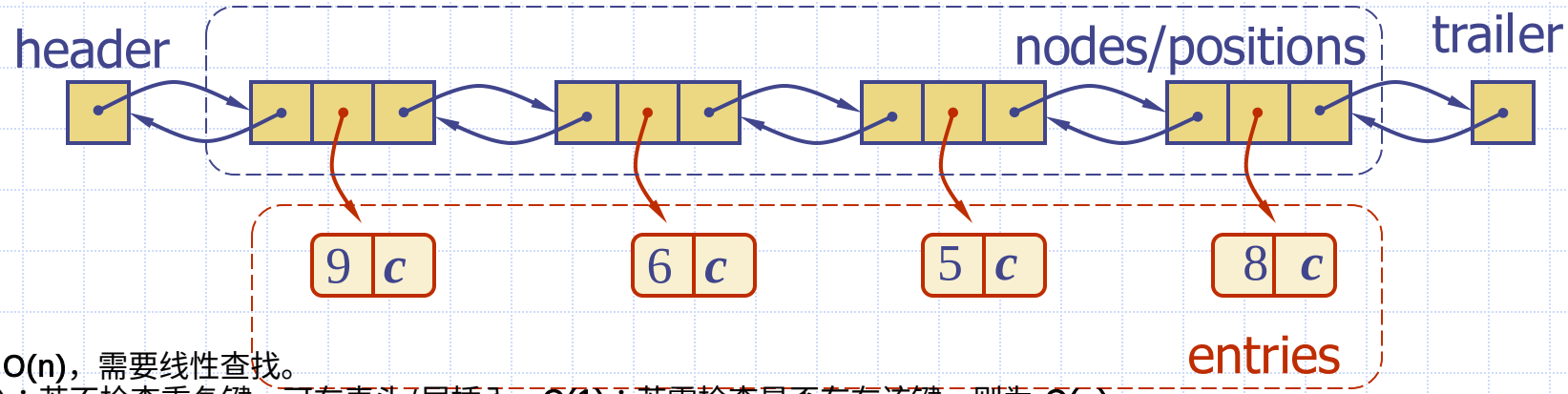
Example

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*

- For example, we can store the items of the map in a list S (based on a doubly linked list), in arbitrary order



get(k)：O(n)，需要线性查找。
put(k, v)：若不检查重复键，可在表头/尾插入，O(1)；若需检查是否存在该键，则为 O(n)。
remove(k)：需遍历链表找键，再删除节点，O(n)。

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 would have taken $O(1)$ time, if we could just insert the new item at the beginning or at the end of the sequence
    - But we have to check if the key occurs in the map, so it is $O(n)$.
  - 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

get(k)：O(n)，需要线性查找。
put(k, v)：若不检查重复键，可在表头/尾插入，O(1)；若需检查是否存在该键，则为 O(n)。
remove(k)：需遍历链表找键，再删除节点，O(n)。