CS310 – Advanced Data Structures and Algorithms

JDK Maps
Spring, 2021

Sets vs. Maps

- Say we had a Set of Student records and we wanted to look up a student (by name or student ID).
- No efficient way to pull out a matching record in a Set.
- We could iterate over the whole Set, and find the match, but that’s O(N), and we want something faster.
- The Map interface allows us to do that. S&W calls it a symbol table or ST.
- Actually S&W’s ST is a class, not an interface, but its API is what we’re talking about.

Maps – Idea

- Given two sets, Domain and Range, with a relation from one to another.
- Like a math function, each domain element has associated with it exactly one range element.
- Two arrows can land on the same range element, but one domain element cannot have two arrows out of it.

Maps – Idea and actions

- The action of following the arrow is often known as a “lookup” action.
- For ex., employee records are looked up by social-security no. and/or by employee name.
- Social security numbers or employee names are the Domain, Employee objects are the Range.
- In programming, Maps (i.e. STs) are lookup-tables.
- We are mapping integers to employee objects or Strings to employee objects.
- Mapping creates a pair of <DomainType,RangeType>, sometimes called an association.
- The DomainType is a key, the RangeType is a value.

Maps – Simple Example, see the set of pairs that define it

- A simple example: mapping scores to letter grades
  - 92 → “A”
  - 79 → “B”
  - 68 → “C”
- Key = score is an integer, the DomainType
- The RangeType is String (could be a character but we want to account for all letter grades like “A-” etc.).
- Each of these lines can be called a “key/value pair”, or just “pair” (or “association”).
- (92, “A”) is a pair of the grade 92 (the key) and the string “A” (the value)
- The whole mapping is the set of these 3 pairs.
- \( M = \{ (92, “A”), (79, “B”), (68, “C”) \} \) – a map as a set of pairs, or associations

Map– a collection but not a Collection

- A mapping is a collection like other collections we are studying, lists, stack, queues, and sets.
- However, in Java a Map has its own interface separate from Collection.
- Note that not every collection of pairs makes a proper map: \( M \) qualifies as a map only if the collection of keys has no duplicates, i.e., constitutes a Set.
- The collection of values can have repetitions, so it is not a Set, just a Collection.
The Map Interface: methods

```java
// Map interface
public interface Map<KeyType,ValueType> extends Serializable {
    // Returns the number of keys in this map.
    int size();
    // Tests if this map contains a given key.
    boolean containsKey(KeyType key);
    // Returns the value associated with the key.
    ValueType get(KeyType key);
    // Returns the original value set to the key; uses map.put().
    ValueType put(KeyType key, ValueType value);
    // Removes the key-value pair from the map.
    void remove(KeyType key);
    // Returns the keys in the map.
    Set<KeyType> keySet();
    // Returns the values in the map. These may be duplicates.
    Set<ValueType> valueSet();
    // Returns a set of Map.Entry objects corresponding to key/value pairs.
    Set<Map.Entry<KeyType,ValueType>> entrySet();
}
```

Actions on Maps

- We can add a key/value pair to a Map, using put(key, value).
- We can look up the associated range element (value) of any given domain element (key) with get(key).
- We can find all the facts in the Map by iterating through the keys and looking up each one with get.
- ...or iterating through all the "entries", i.e., key-value pairs, available by calling entrySet(). It's an iterable...

Map Example, coded and executed

- Like sets, Java supports two main implementations: <b>TreeMap</b> and <b>HashMap</b>. Here we'll use <b>HashMap</b>.
- Note: HashMap, a class, implements Map, an interface

```java
import java.util.Map;
import java.util.HashMap;

public class TestMap1 {
    public static void main(String[] args) {
        Map<Integer,String> map = new HashMap<Integer,String>();
        map.put(92, "A");
        map.put(79, "B");
        map.put(68, "C");
        System.out.println(map);
    }
}
```

Maps – Back to Simple Example

- A simple example: mapping numbers to letter grades
  - 92 → "A"
  - 79 → "B"
  - 68 → "C"
- is an integer and the RangeType is string (could be a character but we don't want to account for all letter grades like "A-") etc.
- Each of these lines can be called a "key/value pair", or just "pair".
- (92, "A") is a pair of the grade 92 (the key) and the string "A" (the value)
- The whole mapping is the set of these 3 pairs.
- Calls to map.entrySet() yield objects of type Map.Entry.<KeyType,ValueType>.

Map Example, showing entrySet

```java
import java.util.Map;
import java.util.Set;
import java.util.HashMap;

public class TestMap1 {
    public static void main(String[] args) {
        Map<Integer,String> map = new HashMap<Integer,String>();
        map.put(92, "A");
        map.put(79, "B");
        map.put(68, "C");
        System.out.println(map);
        System.out.println(map.entrySet());
        for (Map.Entry<Integer,String> e : entries) {
            System.out.println(e);
        }
    }
}
```
Ways of Thinking About Maps

- As holding conversions, like codes to grades, social security number to name.
- As generalized arrays. An array maps 0 to a[0], 1 to a[1], etc., very restricted map
- As math functions: \( y = f(x) \) is a map: each \( x \) to \( f(x) \).
- As a "database" with key lookup: SSN to employee record, ISBN to book record, name to inventory record.

Maps and Sets in S&W, and vs. JDK

- S&W start with "symbol tables", STs for short: these are maps, i.e. key → value containers
- The ST API (page 363) has get, put, and a keys() method that returns an Iterable allowing a scan of contents
- Then later (pp. 496-497) S&W briefly cover sets as STs without values
- This is valid for a simple model of sets, but we will see that JDK Sets can support union, intersection, etc.
- Also JDK Sets support set equality testing across implementations.
- And JDK Maps support map equality defined by set equality of their entrySets

Map Example, showing entrySet, with output

```java
... (code from last slide) 
System.out.println(map);
SetsMap.Entry<Integer,String> e = map.entrySet();
System.out.println(entries);
for (Map.Entry<Integer,String> e : entries) {
    System.out.println(e);
    }
```

This shows that the HashMap overrides Object’s toString() to print out its map contents, and similarly for the Set and Map.Entry classes in use here.

Maps are Sets with a Value attached to each element

- So the elements, i.e. the key objects, need the same equals/hashCode/comareTo treatment as in HashSets/TreeSets
- We need equals and hashCode for key objects of HashMap
- We need equals and compareTo for key objects of TreeMap
- The value objects don’t need any of these methods implemented
- Note: if we ignore the values, a Map provides a ready-made set. This is the approach to Set implementations in S&W.

Set and Map Applications

So far, we’ve looked at “toy” applications: just put things in
- Set or Map, then read them back
Let’s look at S&W for more app ideas
- Set: Dedup, p. 490: read words from a text file, report each once only.
- Set: BlockFilter (new name), pg. 492: read special words from one file, report on their use in a text
- Map: FrequencyCounter, pg. 372: find #occurrences of each word in a text
- Map: LookupCSV, pg. 495: read key-value pairs (one value for each key) from a file, then lookup keys for user.

Set and Map Applications: Converting Code

- We need to know how to convert S&W code to JDK classes
- StdIn.readString, StdIn.isEmpty() : well use Scanner
- S&W HashST: use HashST (or TreeSet), which ISA Set
- S&W ST: use HashMap (or TreeMap), which ISA Map
- Look at the doc on FrequencyCounter.java to see a full example: FrequencyCounter
- The code is in TextMap.java
FrequencyCounter: Demo on running it

- The code is in TestMap.zip: download, unzip it
- Get TestMap directory on your system
- In TestMap dir, compile with javac FrequencyCounter.java
- Run with java FrequencyCounter 8 < tale.txt
- cd SW, to see the S&W version
- Look at README to see how to build and run it there
- Finally, see it in eclipse, get a terminal there

Running Code from cs210

- In cs210, a special VM was used for programming
- So you probably have code examples in this VM
- You can rebuild them outside of the VM
- See the samples in cs210support.zip

Performance of JDK Sets

<table>
<thead>
<tr>
<th>method</th>
<th>JDK HashSet&lt;E&gt;</th>
<th>JDK TreeSet&lt;E&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>size(), isEmpty()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>contains</td>
<td>O(1)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>add</td>
<td>O(1)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>remove</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

* assuming good hash function and large-enough hash table

So best to initialize hash table size to estimated needed size

new HashSet(1000000) for example

Performance of JDK Maps

<table>
<thead>
<tr>
<th>method</th>
<th>JDK HashMap&lt;E&gt;</th>
<th>JDK TreeMap&lt;E&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>size(), isEmpty()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>containsKey</td>
<td>O(1)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>get</td>
<td>O(1)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>put</td>
<td>O(1)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>remove</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>clear</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>keySet</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

* assuming good hash function and large-enough hash table

Can use new HashMap(1000000) for example

JDK Maps provide "views" on their data

- The entrySet() from Map.entrySet() is not just a snapshot of the map contents, but a "view" on the contents, sensitive to changes in the contents.
- Similarly keySet() is a view on the Map's domain, and values() is a view on the Map's range.
- vs. keys() of S&W STs, which are simple snapshots.
- Element adds and deletes to the main contents show in the obtained entrySet.
- In either system (JDK or S&W), updates to an individual element show up in the derived set, since the elements are handled by reference.

ConcurrentModificationException

There are needed limits to this live-view service in terms of interaction between Map operations.

Example
1. Start a scan of M's domain using M.keySet().
2. Delete an entry in M by its key value using remove.
3. Now the iterator from the scan is no longer usable.
Error if you try to use it: ConcurrentModificationException

- All the JDK Collections classes track possibly dangerous combinations of actions and cause an error rather than venture into states of ambiguity or stale data.
- Example using S&W keys(): no error, just stale data returned.
  - Ref page 382, see Queue created and filled, returned, no notification set up between Queue and this ST object.
Avoiding ConcurrentModificationExceptions

Example that doesn’t cause error...
1. Set up a set X of M’s domain using M.keySet().
2. Scan all of X to count it, say (using up that iterator).
3. Delete an entry in M by its key value using remove.
4. Scan X a second time, get a lower count.

Another example that doesn’t cause error...
1. Set up a set X of M’s domain using M.keySet().
2. Iterate through X using iterator I obtained from X...
3. Remove an entry in M using I’s remove method.
4. Finish the iteration.

• You can see that if the iterator is in control of the edit, it can survive it in health.
• We’ll look at real JDK HashMap code in project 2, so you can see how this can be implemented.