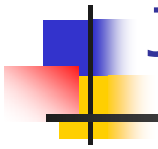
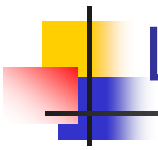


Java Summer 18 Class 15 Notes

Java Collection Framework:
Maps and Sets



Lists, revisited

- Lists are slow for searching
  - `indexOf`, `contains` are slow (O(n))
  - must potentially look at each element of list

```
public int indexOf(Object o) {
  for (int i = 0; i < size(); i++)
    if (get(i).equals(o))
      return i;
  return -1;
}
```

A new collection type: Set

- **set**: an unordered collection with no duplicates
- main purpose of a set is to test objects for membership
- operations are exactly those for `Collection`

- interface `java.util.Set` has the following methods:

```
int size();
boolean isEmpty();
boolean contains(Object e);
boolean add(Object e);
boolean remove(Object e);
Iterator iterator();
```

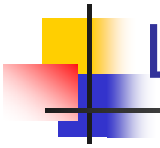
```
boolean containsAll(Collection c);
boolean addAll(Collection c);
boolean removeAll(Collection c);
boolean retainAll(Collection c);
void clear();

Object[] toArray();
Object[] toArray(Object[] a);
```


Set implementations in Java

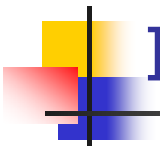
- `Set` is an interface; you can't say `new Set()`
- There are two implementations:
  - `java.util.HashSet` is best for most purposes
  - we won't use the other one: `TreeSet`
  - Java's set implementations have been optimized so that it is very fast to search for elements in them
    - `contains` method runs in constant time! (How?!)

- Preferred: `Set s = new HashSet();`
  Not: `HashSet s = new HashSet();`



Limitations of Sets

- Why are these methods missing from `Set`?
  - `get(int index)`
  - `add(int index, Object o)`
  - `remove(int index)`
- How do we access the elements of the set?
- How do we get a particular element out of the set, such as element 0 or element 7?
- What happens when we print a `Set`? Why does it print what it does?



Iterators for Sets

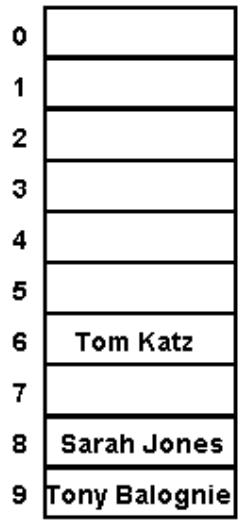
- A set has a method `iterator` to create an iterator over the elements in the set
- The iterator has the usual methods:
  - `boolean hasNext()`
  - `Object next()`
  - `void remove()`

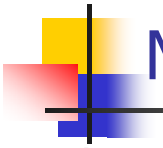
Typical set operations

- Sometimes it is useful to compare sets:
  - **subset**: S1 is a subset of S2 if S2 contains every element from S1.
- Many times it is useful to combine sets in the following ways:
  - **union**: S1 union S2 contains all elements that are in S1 or S2.
  - **intersection**: S1 intersect S2 contains only the elements that are in *both* S1 and S2.
  - **difference**: S1 difference S2 contains the elements that are in S1 that are *not* in S2.
- How could we implement these operations using the methods in Java's Set interface?

How does a HashSet work?

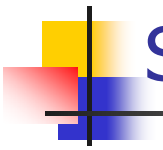
- every object has a reasonably-unique associated number called a *hash code*
  - `public int hashCode()` in class `Object`
- `HashSet` stores its elements in an array a such that a given element o is stored at index `o.hashCode() % array.length`
  - any element in the set must be placed in one exact index of the array
  - searching for this element later, we just have to check that one place to see if it's there (O(1))
    - `"Tom Katz".hashCode() % 10 == 6`
    - `"Sarah Jones".hashCode() % 10 == 8`
    - `"Tony Balognie".hashCode() % 10 == 9`
- you don't need to understand this...





Membership testing in HashSets

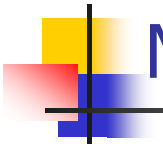
- When testing whether a `HashSet` contains a given object:
  - Java computes the `hashCode` for the given object
  - looks in that index of the `HashSet`'s internal array
    - Java compares the given object with the object in the `HashSet`'s array using `equals`; if they are equal, returns `true`
- Hence, an object will be considered to be in the set only if *both*:
  - It has the same hash code as an element in the set, *and*
  - The equals comparison returns true
- an object that is put into a `HashSet` works best if it has a `public int hashCode()` method defined
  - `String`, `Integer`, `Double`, etc. have this already



Set practice problems

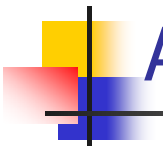
- Modify our Sieve of Eratosthenes to return a `Set` of primes, instead of just printing them.

- Given a `List` of elements or string of many words, determine if it contains any duplicates, using a `Set`. (You can use a `Scanner` to break up a `String` by words.)



Mapping between sets

- sometimes we want to create a mapping between elements of one set and another set
  - example: map people to their phone numbers
    - "Marty Stepp" --> "253-692-4540"
    - "Jenny" --> "253-867-5309"
- How would we do this with a list (or list(s))?
  - A list doesn't map people to phone numbers; it maps `ints` from 0 .. *size* - 1 to objects
  - Could we map some `int` to a person's name, and the same `int` to the person's phone number?
  - How would we find a phone number, given the person's name? Is this a good solution?



A new collection: Map

- **map**: an unordered collection that associates a collection of element values with a set of keys so that elements they can be found very quickly (O(1)!)
  - Each key can appear at most once (no duplicate keys)
  - A key maps to at most one value
  - the main operations:
    - **put**(key, value)
      "Map this key to that value."
    - **get**(key)
      "What value, if any, does this key map to?"
  - maps are also called:
    - hashes or hash tables
    - dictionaries
    - associative arrays

Java's Map interface

```
public interface Map {
  Object put(Object key, Object value);
  Object get(Object key);
  Object remove(Object key);
  boolean containsKey(Object key);
  boolean containsValue(Object value);
  int size();
  boolean isEmpty();

  void putAll(Map map);
  void clear();

  Set keySet();
  Collection values();
}
```

Basic ops

Bulk ops

Collection views

Map implementations in Java

- `Map` is an interface; you can't say `new Map()`
- There are two implementations:
  - `java.util.HashMap` is best for most purposes
  - we won't use the other one: `TreeMap`

- Preferred: `Map m = new HashMap();`
  Not: `HashMap m = new HashMap();`

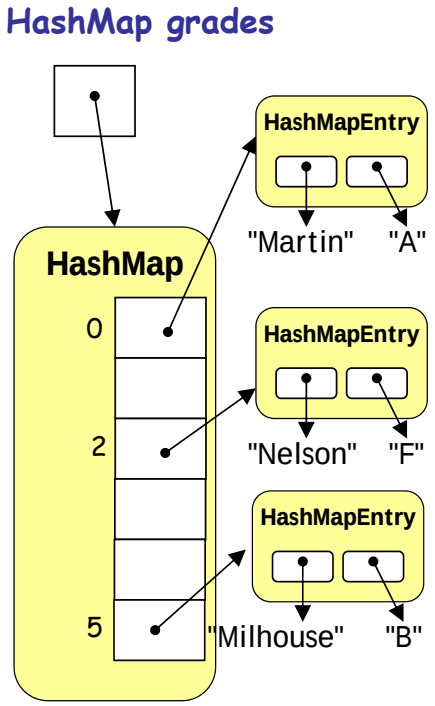
HashMap example

```
HashMap grades = new HashMap();
grades.put("Martin", "A");
grades.put("Nelson", "F");
grades.put("Milhouse", "B");

// What grade did they get?
System.out.println(
  grades.get("Nelson"));
System.out.println(
  grades.get("Martin"));

grades.put("Nelson", "W");
grades.remove("Martin");

System.out.println(
  grades.get("Nelson"));
System.out.println(
  grades.get("Martin"));
```



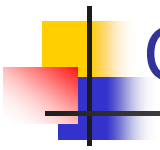
Map example

```
public class Birthday {
    public static void main(String[] args){
        Map m = new HashMap();
        m.put("Newton", new Integer(1642));
        m.put("Darwin", new Integer(1809));
        System.out.println(m);
    }
}

Output:
{Darwin=1809, Newton=1642}
```

Some Map methods in detail

- `public Object get(Object key)`
  - returns the value at the specified `key`, or `null` if the key is not in the map (constant time)
- `public boolean containsKey(Object key)`
  - returns `true` if the map contains a mapping for the specified key (constant time)
- `public boolean containsValue(Object val)`
  - returns `true` if the map contains the specified object as a value
  - this method is *not* constant-time O(1) ... why not?



Collection views

- A map itself is not regarded as a collection
  - `Map` does not implement `Collection` interface
  - although, in theory, it could be seen as a collection of pairs, or a relation in discrete math terminology
- Instead collection *views* of a map may be obtained
  - Set of its keys
  - Collection of its values (not a set... why?)

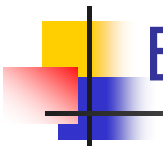
Iterators and Maps

- `Map` interface has no iterator method; you can't get an Iterator directly
- must first call either
  - `keySet()` returns a `Set` of all the keys in this `Map`
  - `values()` returns a `Collection` of all the values in this `Map`
- then call `iterator()` on the key set or values
  - Examples:

```
Iterator keyItr = grades.keySet().iterator();
Iterator elementItr = grades.values().iterator();
```

  - If you really want the keys or element values in a more familiar collection such as an `ArrayList`, use the ArrayList constructor that takes a `Collection` as its argument

```
ArrayList elements = new ArrayList(grades.values());
```

Examining all elements

- Usually iterate by getting the set of keys, and iterating over that

```
Set keys = m.keySet();
Iterator itr = keys.iterator();
while (itr.hasNext()) {
    Object key = itr.next();
    System.out.println(key + "=>" +
                       m.get(key));
}
```

```
Output:
Darwin => 1809
Newton => 1642
```

Map practice problems

- Write code to invert a `Map`; that is, to make the values the keys and make the keys the values.

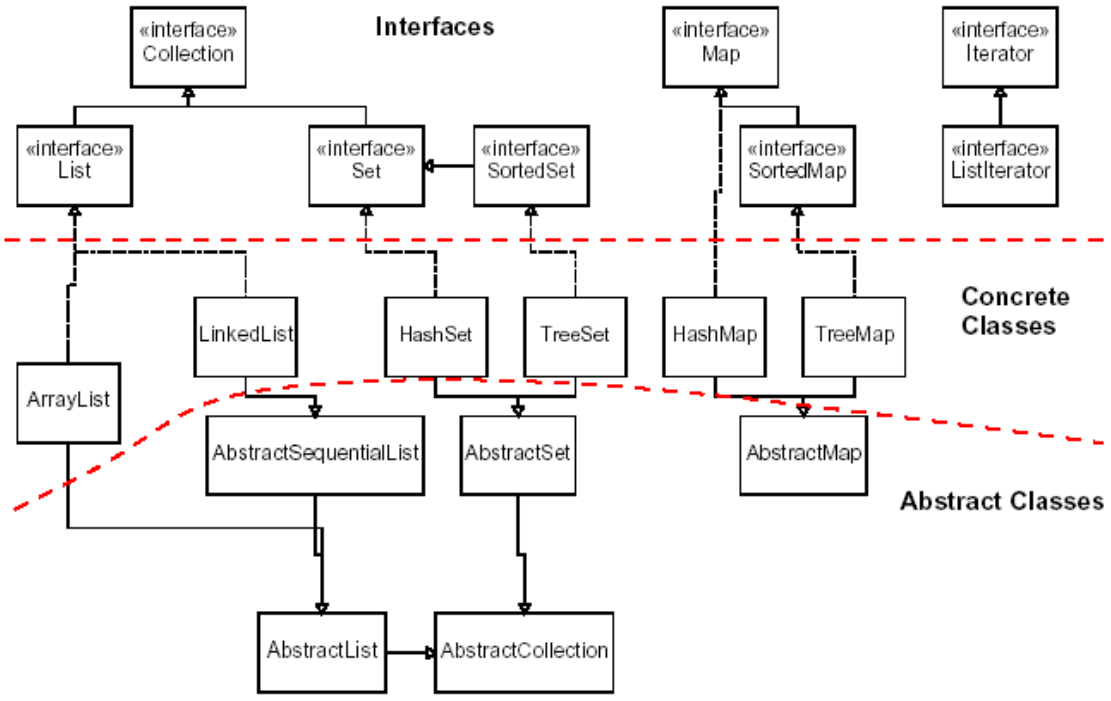
```
Map byName = new HashMap();
byName.put("Darwin", "748-2797");
byName.put("Newton", "748-9901");

Map byPhone = new HashMap();
// ... your code here!
System.out.println(byPhone);

Output:
{748-2797=Darwin, 748-9901=Newton}
```

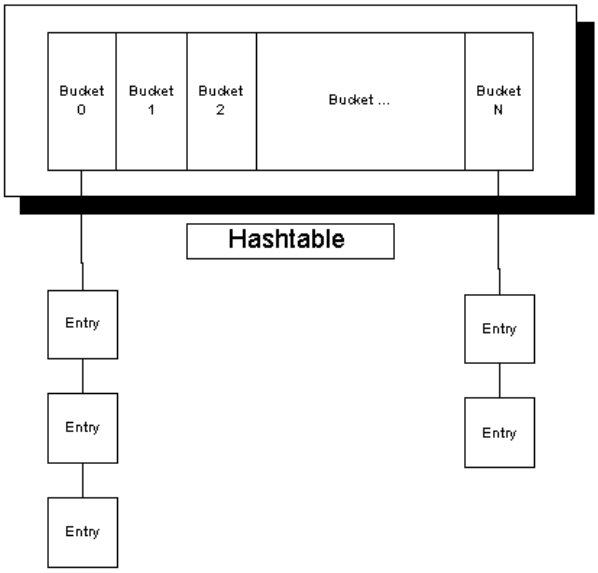
- Write a program to count words in a text file, using a hash map to store the number of occurrences of each word.

Java Collections Framework



Compound collections

- Collections can be nested to represent more complex data
- example: A person can have one or many phone numbers
  - want to be able to quickly find all of a person's phone numbers, given their name
- implement this example as a HashMap of Lists
  - keys are Strings (names)
  - values are Lists (e.g ArrayList) of Strings, where each String is one phone number



Compound collection code 1

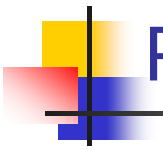
```
// map names to list of phone numbers
Map m = new HashMap();
m.put("Marty", new ArrayList());
...
ArrayList list = m.get("Marty");
list.add("253-692-4540");
...
list = m.get("Marty");
list.add("206-949-0504");
System.out.println(list);
```

[253-692-4540, 206-949-0504]

Compound collection code 2

```
// map names to set of friends
Map m = new HashMap();
m.put("Marty", new HashSet());
...
Set set = m.get("Marty");
set.add("James");
...
set = m.get("Marty");
set.add("Mike");
System.out.println(set);
if (set.contains("James"))
  System.out.println("James is my friend");
```

```
{Mike, James}
James is my friend
```

References

- Koffman/Wolfgang Ch. 9, pp. 453-464
- *The Java Tutorial: Collections.* http://java.sun.com/docs/books/tutorial/collections/index.html
- *JavaCaps Online Tutorial: java.util package.* http://www.javacaps.com/scjp_util.html