

Tutorial 13

December 5-6, 2005

Today's Topics

- Laptops!!
- Hashing
- Collections Framework
- Problem Set 9 Q/A

Laptops!

- Return laptops @ final exam.
- You must bring all your accessories too.
- Don't forget! You don't have time to run back and get it during the exam.

Review of Big-O

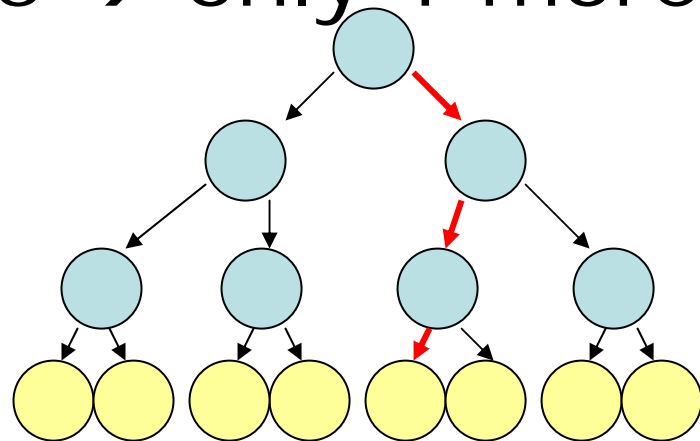
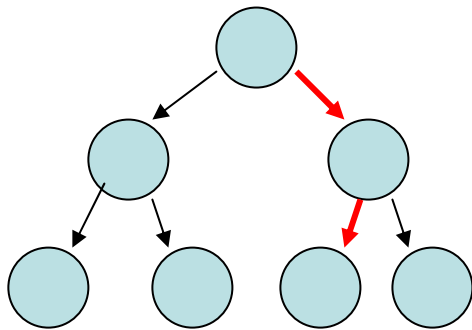
- “Big-O” notation $O()$ is a measure of running time.
- The argument tells you what the running time is proportional to.

Review of Big-O

- $O(n)$: running time is proportional to n , the number of items being processed.
- Example: linked list lookup. If you have a list with $2x$ as many links, it'll take $2x$ as long to look through the list.

Review of Big-O

- $O(\log n)$: running time is proportional to the log of n (usually base 2).
- Example: binary search tree lookup.
2x as many nodes \rightarrow only 1 more step!

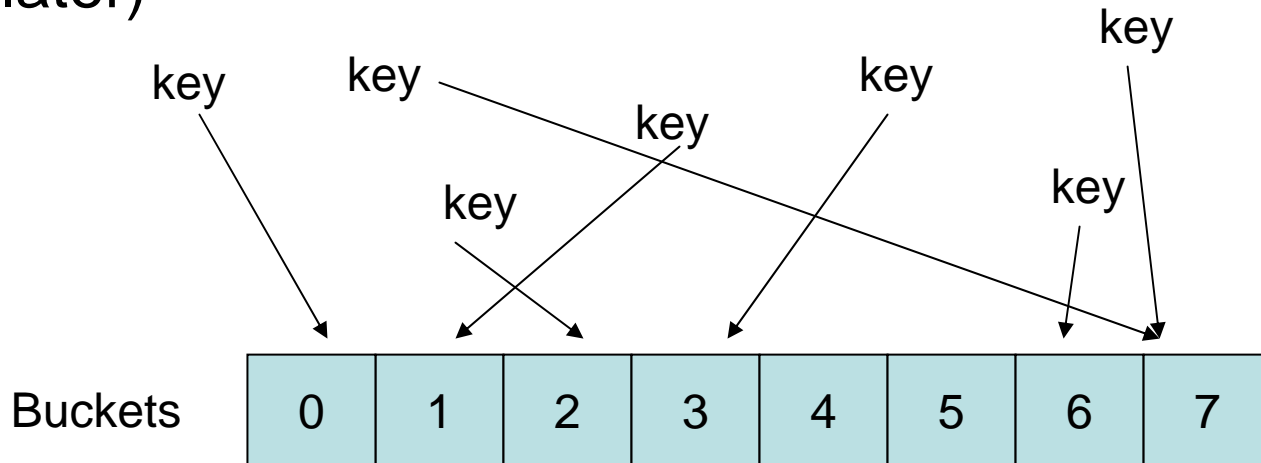


Review of Big-O

- $O(1)$: running time is proportional to 1 (a constant).
- Example: getting the n 'th index of an array. It takes no more time to read `array[10000]` than it does to read `array[23]`.
- Hashing is concerned entirely with this last scenario because it's so fast!

Hashing

- In hashing we call the underlying array slots “buckets”
- We want to be able to use arbitrary “keys” to access the buckets.
- We can convert keys to indices using a hash method. This means more than one key might lead to the same bucket, but each key will only be associated with one bucket. (more on this later)



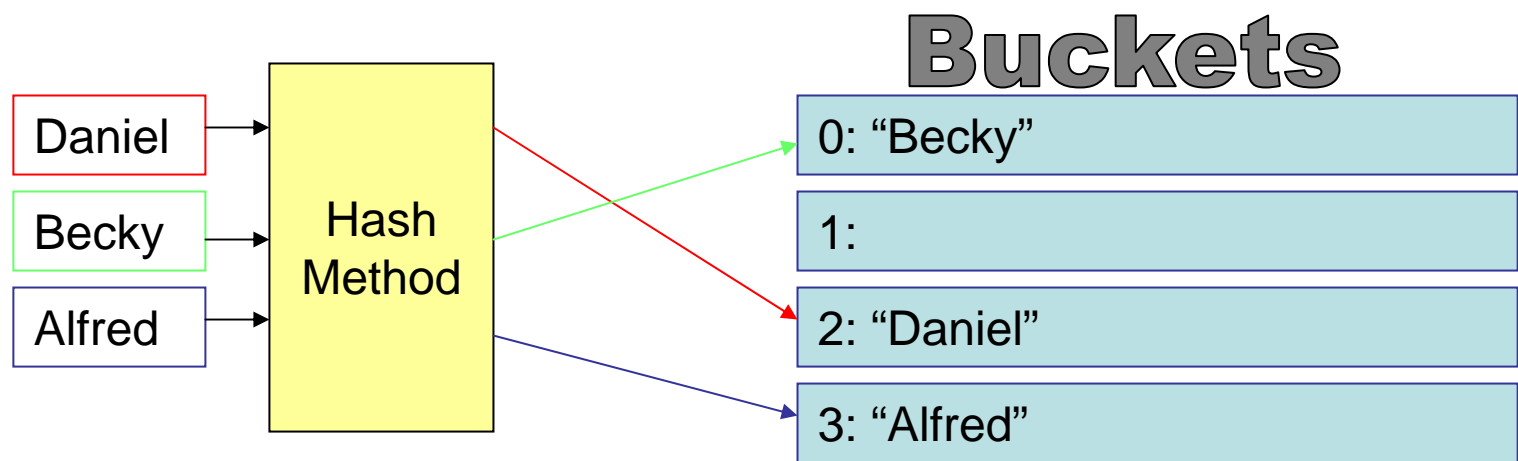
Hashing

- We are looking at two forms of this hash based data structure:
 - Hash tables.
 - Hash maps.

Hashing – hash tables

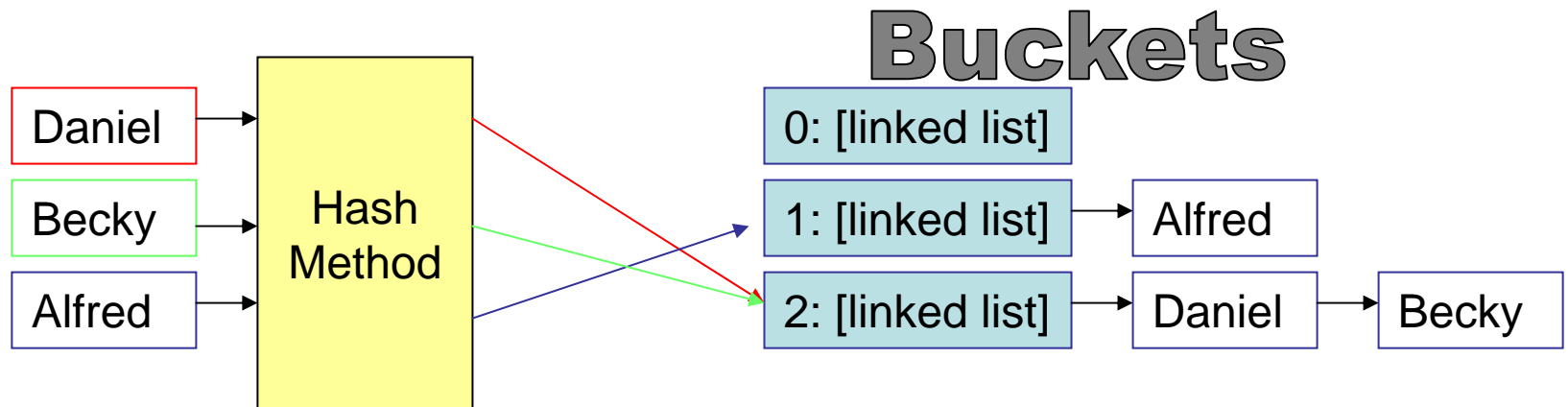
- In hash tables the key is also the data we're storing.
- Example: a hash table of the names of all the students in the class. The keys are the names, and that's all the data we're interested in. We run the hash method on the keys, and store the keys at the index given by the hash method.

Hashing – hash tables



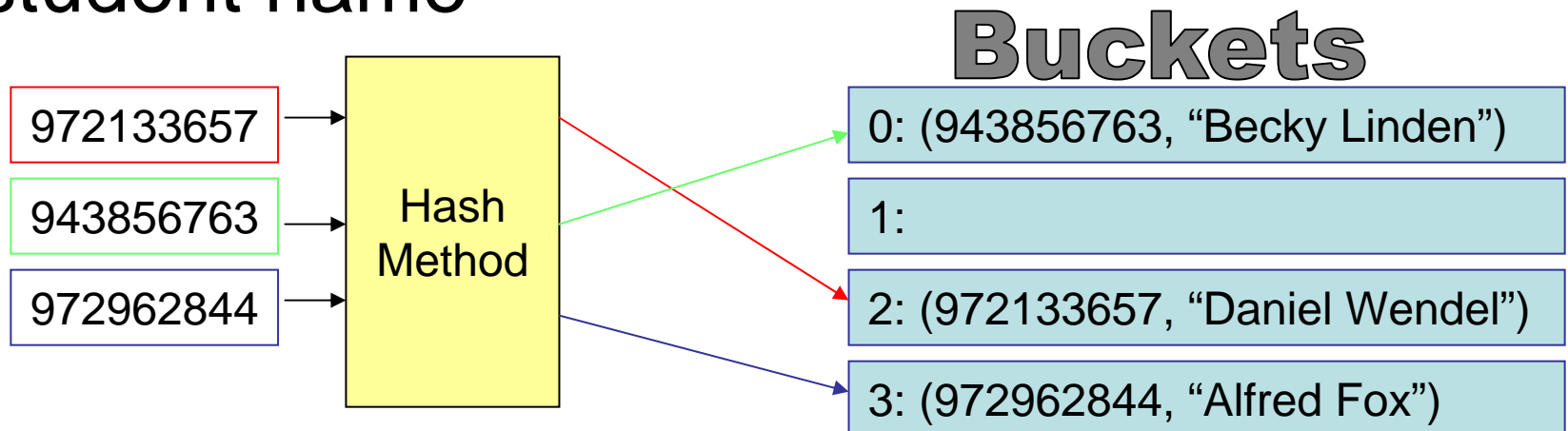
Hashing – handling collisions

- To handle the case where more than one key leads to the same bucket, make each bucket actually be a linked list so it can store more than one key.



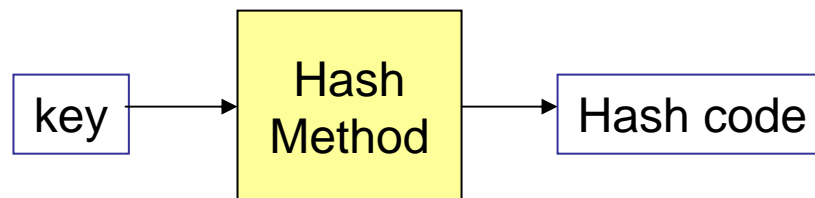
Hashing – Hash maps

- In hash maps, each key is associated with some other data, called the “value”. We store the (key, value) pair in the bucket rather than just the key.
- Example: key is student ID #, value is student name



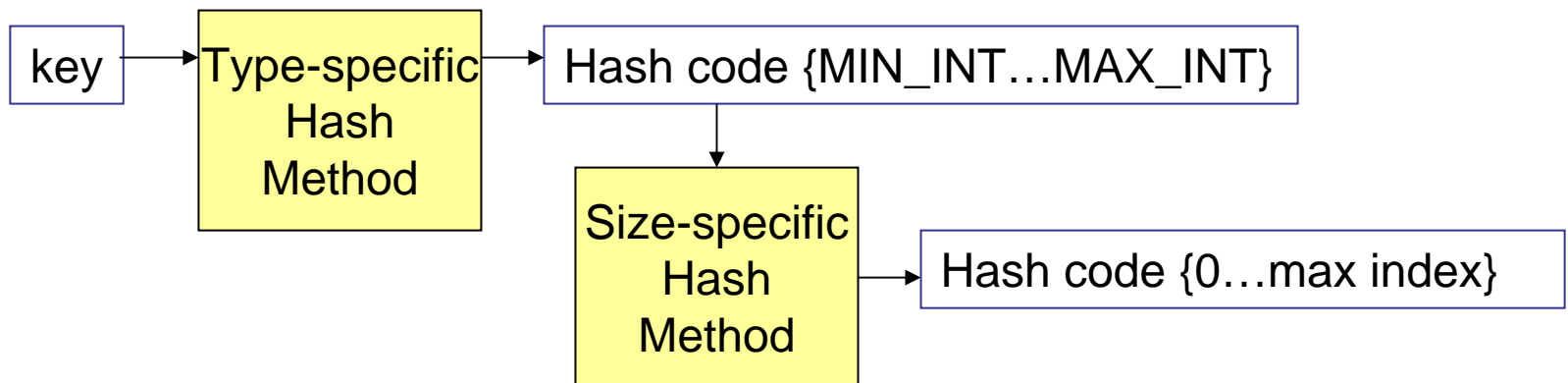
Hashing – Hash methods

- So how do we make this mystical hash method that converts keys to indices?
- A hash method takes in a key, and does some tricky math operations on the key to produce a hash code, or “hash”.
- The same operations are done on each key, but the produced indices are different for different keys.



Hashing – hash methods

- Typically this is done in two steps.
 1. Use a hash method specific to the key's data type to produce a hash code integer
 2. Use a different hash method that takes the integer in and produces an index for a given size array.

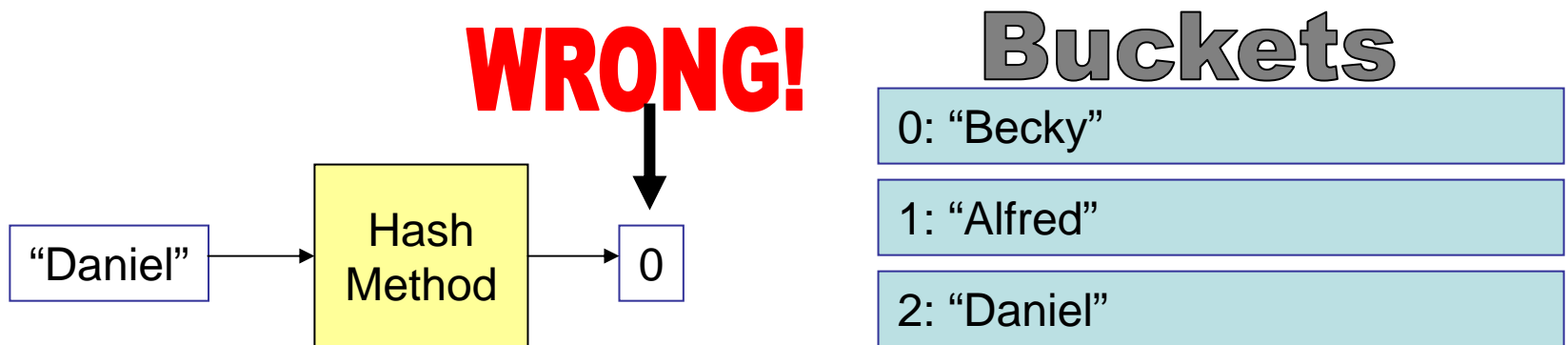


Hashing – hash methods

- There are many different good hash methods, but they all have these properties:
 1. equivalent keys produce equivalent hashes
 2. slightly different keys produce radically different hashes
 3. the hashes produced for a set of several different keys are evenly spread throughout the whole output range.
- Why these properties? Well, let's look at what goes wrong if these properties DON'T hold.

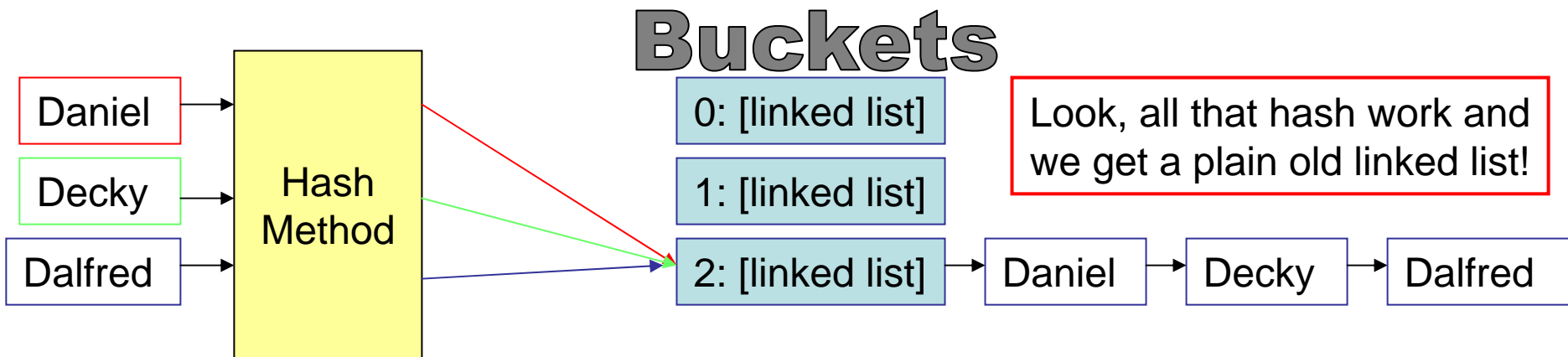
Hashing – hash methods

- Equivalent keys produce equivalent hashes
- I want to check to see if “Daniel” is in my hash table of student names, so I send “Daniel” through the hash method and look at the resulting bucket. If it gives me the wrong index, I won’t find “Daniel” so I’ll think it hasn’t been added to the table.



Hashing – hash methods

- Slightly different keys produce radically different hashes
- With the names hash table, if we only added names starting with D and the hash method only used the first letter to get its value, we'd turn our hash table into a linked list!



Hashing – hash methods

- The hashes produced for a set of several different keys are evenly spread throughout the whole output range.
- If hash codes are clustered, turns the hash table into a small group of linked lists, leaving the rest of the buckets empty.
- If some hash codes are never produced, those buckets will stay empty!

Hashing – good hash methods

String Class hashCode(), 2

```
public int hashCode() {
    int h = hash;
    if (h == 0) {
        int off = offset;
        char val[] = value;
        int len = count;

        for (int i = 0; i < len; i++)
            h = 31*h + val[off++];
        hash = h;
    }
    return h;
}
```

Integer Hashing

A good method to hash an integer (including our hash codes) multiplies the integer by a number, A , $0 < A < 1$, extracts the fractional part, multiplies by the number of table slots, m , and truncates to an integer. In Java, if n is the integer to be rehashed, this becomes

```
private int hashCode( int n ) {
    double t = Math.abs( n ) * A;
    return ( (int) (( t - (int)t ) * m ) );
}
```

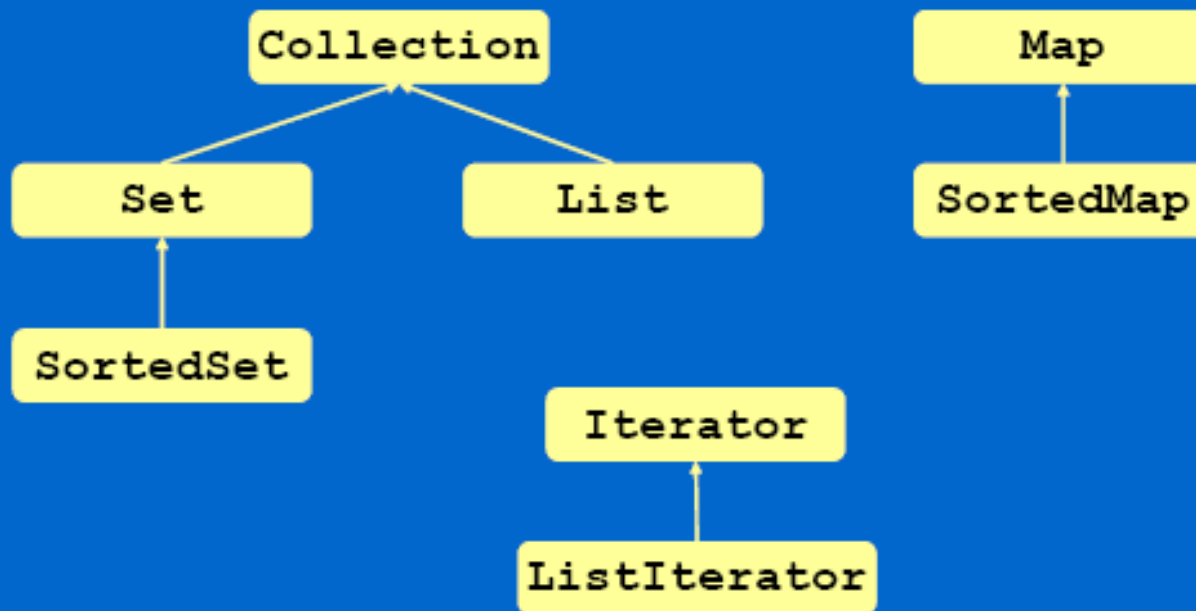
Unintuitively, certain values of A seem to work much better than others. The literature suggests that the reciprocal of the golden ratio, $(\text{sqrt}(5.0) - 1.0) / 2.0$ works particularly well.

Hashing – other applications

- Hashing can be used for other things besides creating hash tables.
- Another important application is in security.
- If you send the hash of the contents of a file with a file, the person who receives it can run the same hash method on the contents and compare the result to the hash code you provided.
- If it doesn't match, the file was corrupted during transport.

Collections framework

Collections Interfaces, 1



Collections framework - interfaces

- Collection: is the most basic interface; it has the functionality of an unordered list, aka a *multiset*, a set that doesn't not check for duplicates.
- Set: adds set semantics; that is, it won't allow duplicate members.
- List: adds list semantics; that is, a sense of order and position
- SortedSet: adds order to set semantics; no duplicates, with a sense of order and position

Collections framework - interfaces

- Map: the basic interface for data structures that map keys to values; it uses an inner class called an Entry
- SortedMap: a map of ordered keys with values; our binary search tree is a good example
- Iterator: our Iterator except that it is *fail fast*; it throws a ConcurrentModificationException if you use an instance after the underlying Collection has been modified.
- ListIterator: a bidirectional iterator.

Collections framework - implementations

- *Resizable Array*: similar to the technique used for our Stack implementation.
- *Linked List*: uses a doubly, not singly, linked list.
- *Hash Table*: very similar to our implementation except that it will grow the number of slots once the load factor passes a value that can be set in the constructor.
- *Balanced Tree*: similar to our binary search tree implementation but based on the more sophisticated Red-Black tree that rebalances the tree after some operations.

Collections framework - implementations

| | | <i>Implementations</i> | | | |
|-------------------------|-----------------------|------------------------|------------------------|----------------|-------------------------|
| | | Hash Table | Resizable Array | Balanced Tree | Linked List |
| <i>Inter- faces</i> | <i>List</i> | | <i>Array- List</i> | | <i>Linked- List</i> |
| | <i>Set</i> | <i>HashSet</i> | | | |
| | <i>Sorted Set</i> | | | <i>TreeSet</i> | |
| | <i>Map</i> | <i>HashMap</i> | | | |
| | <i>Sorted Map</i> | | | <i>TreeMap</i> | |

Collections framework

- The Collections Framework gives you a bunch of tools to work with.
- However, it's important to choose the right tool.
- Choosing a bad implementation for your task might make your program run very inefficiently.

Collections framework

- Choose an implementation if...
 - You need adds and lookups to be really fast but it's okay for removes to be slow (online directory, for example).
 - You need fast adds and removes but lookups can be slow (queues).
- Sometimes space is more important than speed (on phones, PDAs).
- Sometimes speed is crucial (control systems in airplanes)

Problem Set 9

- BoxField – derivative of JTextField that allows only one character.
- Questions?
- How is the solve method coming?