A hash table for a set of Objects

Functions equals and hashCode

An implementation of a set needs a way to test equality of elements, so that an element is not put into the set twice. For now, we rely on function equals in class Object, so that a set may contain only objects that do not override function equals. Thus, x.equals(y) is true iff x and y are the same object.

The implementation will also use function hashCode() in class Object. For object x, x.hashCode() returns the address in memory of x.

The basic data structure used to implement the set

We now describe the basic data structure. We have an array b, shown here with 9 elements, each element of which is declared to be a LinkedList. So, each element of b is either null or an object of class LinkedList. In this context, array elements b[h] are called *buckets*.

The set being implemented consists of the elements in all the LinkedLists —note that no element appears more than once in all these LinkedLists. That's all there is to it!

Method add

We now develop most of the code for add, showing the addition of object e1 to the empty set as an example. Remember: e1 is actually a pointer to an object, not the object itself.

Method add(e) returns false if the element is already in the set and true if e was not in the set and was added. The big question is:

To which of the 9 buckets b[0..8] should e be added to?

The answer: Calculate the remainder h when e.hashCode() is divided by the length of b and use only bucket b[h]. Integer h, based on the address where e is stored in memory, is rather random. This randomness will give us, when we are done, O(1) expected time for add and remove.

If b[h] is **null**, create a new empty LinkedList and store it in b[h]. Next, if LinkedList b[h] contains e, e is already in the set and false is returned. Finally, if e was not in b[h], add it.

Consider attempting to add object e1 again, calling add(e1). Again, h is set to 5. Element b[h] is not **null**. This time the LinkedList contains e1 and false is returned.

We show what the data structure might look like with 6 elements, in buckets b[2], b[5], and b[7].

The next video describes what to do when the set becomes "too full", that is, when the linked lists get too long.

Here is method add —but it is not yet finished.

```
/** If e is not in the set, add it. Return true iff it was added */
public boolean add(Object e) {
   int h= Math.abs(e.hashCode() % b.length);

   // investigate only bucket b[h]
   if (b[h] == null) b[h]= new LinkedList();
   if (b[h].contains(e)) return false;
   return b[h].add(e);   // in order to achieve O(1) expected time, this line must be changed
}
```