Hash codes and indices

- Two step process:
  1. Hash a key into an `int` ("hash code")
  2. Turn a hash code into an array index ("index derivation")
     - Depends on array length!
- `Object` defines a `hashCode()` method
  - Any Java object can be used as a key
  - Implementer must ensure hash code is consistent with equality
    - If overriding `equals()`, *must* override `hashCode()` too!
- Keys should be *immutable*
  - If hash code changes, entries will be at the wrong index
  - Ex) Lists are bad keys!

Step 1: Implementing `hashCode()`

- Goal: two non-equal objects should be *unlikely* to share a hash code
  - Should depend on *all* of an object's state
  - Should depend on *ordering* of any sequential state (e.g. arrays)
  - Should span whole range of integers
- `Objects.hash()`, `Arrays.hashCode()` can help
- When analyzing performance, we will assume `hashCode()` is O(1)
  - i.e. independent of the parameter we're analyzing.
  - Usually we want to analyze how many entries there are, not about the size of the entries.
  - Long strings, data tables would not make performant keys

Step 2: Deriving an index

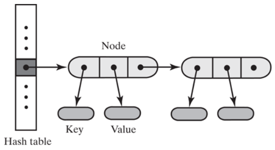
- h(“Hopper”) -> -95141326; Now what?
- Need an index between 0 and array length
- Solution: compute the remainder
  - index = abs(hash % a.length)

Index	Element
0	null
1	null
2	null
3	(Turing, 1912-06-23)
4	null
5	(Johnson, 1918-08-26)
6	(Hopper, 1906-12-09)
7	(von Neumann, 1903-12-28)

Collision resolution approaches

Chaining

- Treat array elements as “buckets” storing a *collection* of entries (e.g. a linked list)
- Finding the right bucket is O(1), but searching it will be slower



Probing

- Array elements point directly to entries
- If desired element is occupied, pick the next element to try according to a probing sequence

Exercise: Chaining example

0	
1	
2	
3	
4	
5	
6	
7	

Informational table

Key	Hash code	Index (%8)	Value
Jenny	126	6	x5309
Eddie	97	1	x7766
Brenda	86	6	x5635
Jack	255	7	x5555
Stacy	118	6	x7666

Load factor

$$\lambda = \frac{\text{number of elements}}{\text{number of buckets}}$$

- May be >1 for chaining (but not for probing)
- Expected cost of lookup with chaining is O(λ)
  - For probing, see DSAJ
- Is that good?
  - If array size is fixed, then λ is O(*N*)
  - If array size is proportional to *N*, then λ is O(1)
- Must use a **dynamic array** for good performance

Exercise: Linear probing example

0	
1	
2	
3	
4	
5	
6	
7	

Key	Hash code	Index (%8)
Jenny	126	6
Eddie	97	1
Brenda	86	6
Jack	255	7
Stacy	118	6

Linear Probing Exercise

Key	Hash code	Index (%8)
Jenny	126	6
Eddie	97	1
Brenda	86	6
Jack	255	7
Stacy	118	6

0	Jack
1	Eddie
2	Stacy
3	
4	
5	
6	Jenny
7	Brenda

Remove Brenda, then ask whether the set contains Stacy.

1. What *should* happen?
2. What *will* happen?