

# Interfaces finalized

# Hashing

Ch. 6

CLR (see class resources page)

# Administrivia

---

- PI Update I (PI v. 1.1) released today
  - On the web site; no paper version
- Prof. Lane out of town, Jan 27-Jan 30
  - May have partial email access
  - Jason will also track cs35 I mail list

# Progress meter

- Milestone 1 due: Jan 31, 3:00 PM (next Mon)
  - 5 days left
- At this point, you should have implemented:
  - Test program
  - **At least** `put()`, `get()`, `containsKey()`, `remove()`, `size()`, `isEmpty()`
- **By end of Friday:**
  - **Complete** `MondoHashMap`
- **By Mon afternoon:**
  - Testing and documentation

A decorative L-shaped line consisting of a vertical line on the left and a horizontal line on top, both in a light gray color, framing the top-left corner of the slide.

# Thoughts on Hashing

# A hash table is...

- Just a data structure that's used to establish a *mapping* (in the mathematical sense) between arbitrary objects
- Every key is assigned (*mapped to*) a single value
- Each key appears only once
- A value may appear more than once
- Not to be confused with `java.util.Map`
- An interface that specifies methods that a program could call to get/set/query key/value mappings
- Hash table is one (but not only) way to make a `Map`

# Examples of mappings

- $37 \rightarrow 6.0827$  (`sqrt(37)`)
- `123-45-6789`  $\rightarrow$  `studentRecord("J. Student")`
- `"ACM"`  $\rightarrow$  `"http://acm.cs.unm.edu/"`
- `"Lane"`  $\rightarrow$  `{ "http://www.cs.unm.edu/~terran/", "http://www.cs.unm.edu/people/faculty/", ... }`

# If all keys were integers...

- If all keys were (small, nonnegative) integers
- Mapping could be made w/ a simple table

```
Object[] mapTab=new Object[1000];  
mapTab[key]=value;
```
- But how do you use an arbitrary object (String, Date, StudentRecord, etc.) as a key?
- Could use the bitwise representation of key object as an integer index
  - Ick!
- Could assign each key object a unique index value
  - Doable, but painful

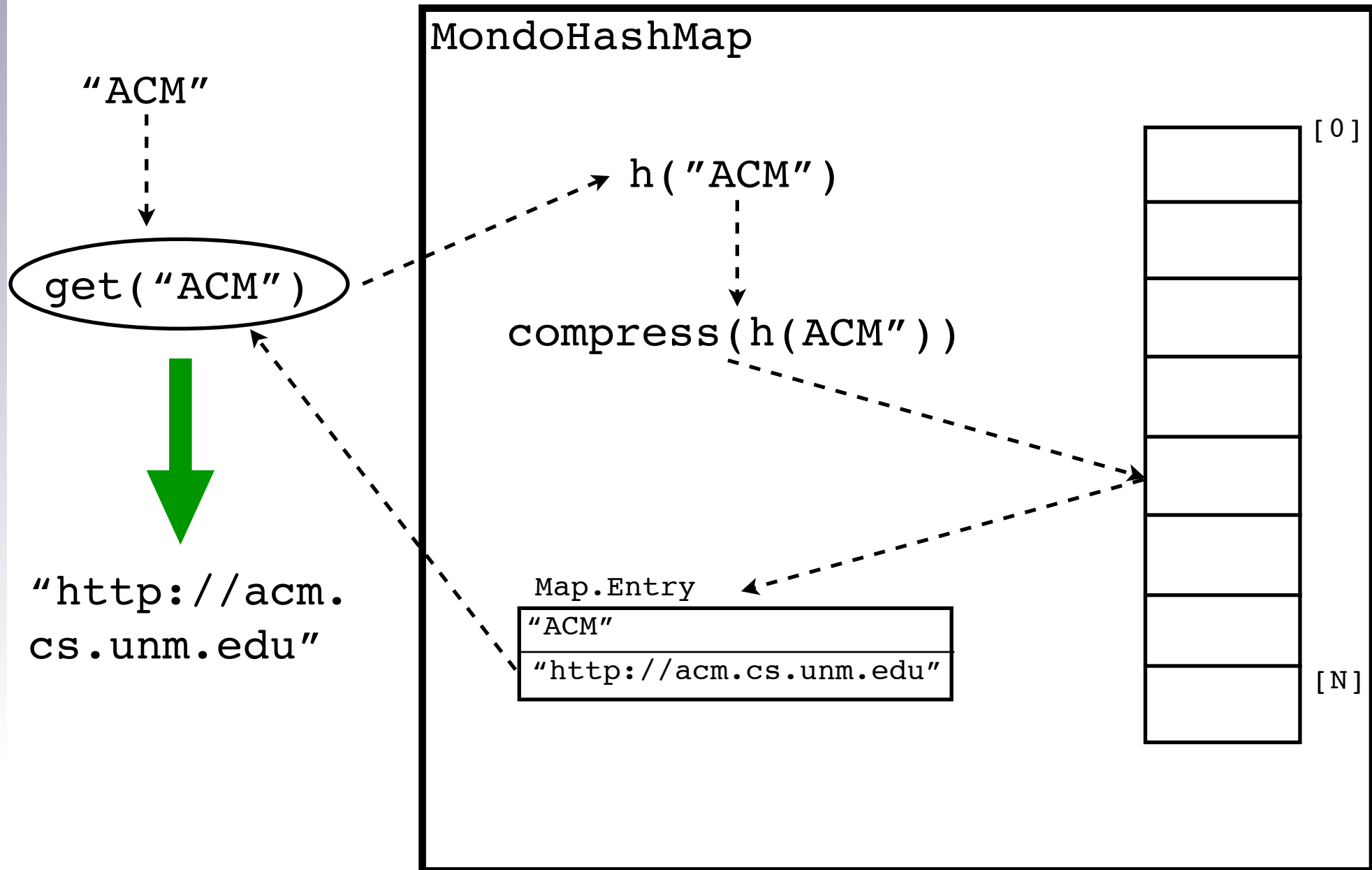
# Desiderata

- An object that gives quick lookup for *arbitrary* key objects
- Doesn't require vast space
- Doesn't need to know anything about the key or value object types (e.g., what's a good hash code for them)

# Hash tables and codes

- Answer: introduce an intermediate step
- *Hash function*: turns an arbitrary key into a *hash code* (integer)
- Use hash code to look up key/val pair in table
- `table[h(key)] = <key, value>`
- Or, in Java:
  - `table[key.hashCode()] = new Entry(key, value);`

# The "big picture"



# Easy enough, in theory...

- Practical gotchas
- How does the `compress()` function work?
- Resizing the hash table
- Handling collisions
  - What if `!k1.equals(k2)`, but  
`compress(k1.hashCode()) == compress(k2.hashCode)`?
- Handling `remove()`
- `keySet()`, `values()`, and `entrySet()`

# Design exercise

- **Given:** A `LinkedList` class, with methods `size()`, `prepend()`, `append()`, `first()`, `last()`, `iterator()`, `removeFirst()`, and `removeLast()`
- **Design:** A sub-class, `StackGenList`, that supports a method `getStackView()` which returns a `Stack` object that, in turn, supports `push()`, `pop()`, `topElement()`, and `depth()`
- **Constraints:**
  - You *do not* have access to the internals of the `LinkedList` (it has no protected members)
  - You *may not* copy any of the `LinkedList` data
  - The `Stack` view and the `LinkedList` must always remain synchronized