"Dictionaries"

programmer: use Hashmap

language designer: Build Hashmap

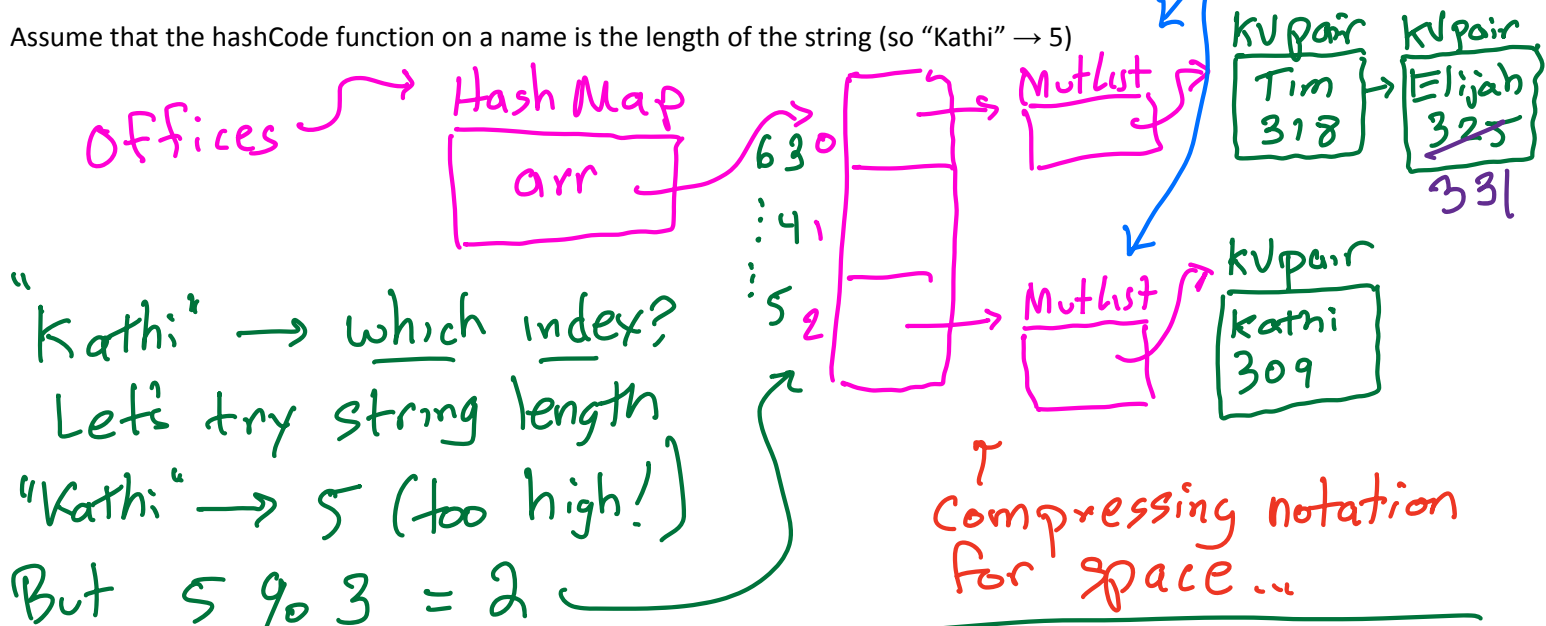
HashMaps Review

```
// create a hashmap with three slots in the underlying array
HashMap<String, Integer> offices = new HashMap<>(3);
offices.put("Kathi", 309);
offices.put("Elijah", 325);
offices.put("Elijah", 331);
offices.put("Tim", 318);
```

← updated — can't have duplicate keys

[I like calling the sub-lists "buckets" :)]

Assume that the hashCode function on a name is the length of the string (so "Kathi" → 5)



"Kathi" → which index?

Let's try string length

"Kathi" → 5 (too high!)

But 5 % 3 = 2

Compressing notation for space...

STEP 1: convert the key object to an int.

STEP 2: "wrap around" the array size

hashCode() method of objects:

new Course("CSCI", 200).hashCode()

HashMap RunTime

How can we recover O(1) runtime?

Q: What is the runtime of get(0) on a linked list?

A: Constant — the length doesn't matter.

Q: What is the runtime of get(100000)?

A: Also constant. We have an upper bound, and the list length can only make it smaller.

So: if we can limit how big the buckets can get, we recover O(1)...

... That does mean the HashMap has to grow dynamically, like array-lists. But you will Not need to do resizing on homework 3.

M4	CIT444
M8	CIT501
T1	CIT501
T4	CIT267
T7	CIT501

Understanding HashCodes

Assume we want to use a hashmap to store this mapping from lab times to rooms (M4 is a shorthand for "Monday 4-6")

We also need keys to be evenly distributed between buckets. (Length doesn't do this.)

ascii("M") + ascii("4")

77 + 52 = 129

(% array size)

ASCII Codes

A	65		1	49
M	77		4	52
T	84		8	56

Java's hashCode() on Strings is even better than this...

You'll use hashCode() on Homework 3 to use objects as keys.

Handling Errors (Exceptions)

```
public class CourseData {
   LinkedList<String> allStudents;

   public CourseData() { this.allStudents = new LinkedList<>(); }

   // addStudent adds student to allStudents if they are not already there
   public void addStudent(String student) {
       if (this.allStudents.contains(student))
           throw new RuntimeException("Student " + student + " already in course");
       this.allStudents.add(student);
   }
}
```

throw new StudentEnrolledException(student)

we define this as a new class that extends Exception. (see the notes.)

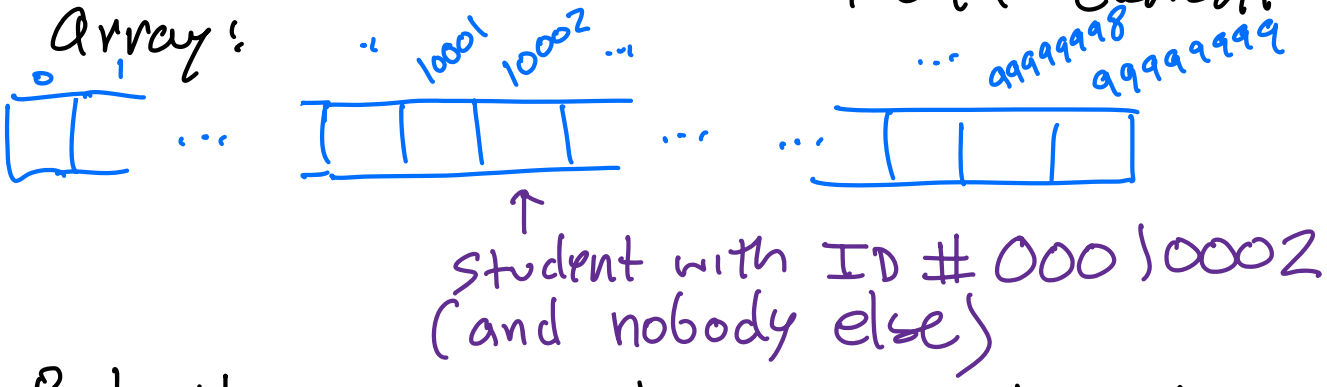
```
public class Registration {
   HashMap<String, CourseData> allCourses;

   // constructor sets up the hashmap with an empty
   // CourseData object for each course name
   public Registration(String[] coursenames) {
       this.allCourses = new HashMap<>();
       for (String name : Arrays.asList(coursenames)) {
           this.allCourses.put(name, new CourseData());
       }
   }

   // add student to given course
   public void enroll(String student, String coursename) {

       allCourses.get(coursename).addStudent(student);
   }
}
```

try{ } catch(StudentEnrolledException e){ System.out.println("student already enrolled"); }

```
public class Main {
   public static void main(String[] args) {
       String[] fall25Courses = {"CS111", "CS17", "CS15"};
       Registration fall25 = new Registration(fall25Courses);
       System.out.println(fall25);
       fall25.enroll("Priya", "CS111");
       System.out.println(fall25);
   }
}
```

Conceptual example from class...

Brown ID #s are 8 digits, that is 0 — 99,999,999. 100 million possibilities! So we could guarantee constant-time search for students by ID if we create a 100M-element array:



↑ student with ID # 00010002 (and nobody else)

But there are nowhere near 100M students! So nearly all of that array will be wasted.

But: we don't know ahead of time which ID #s we will see and have to store! So the "key space" is still enormous.

Hash Maps are the answer to: how can we store a realistic amount of ARBITRARY ID#s in a much smaller array.

To do that, we need handle collisions somehow — hence the "buckets".

To keep them under some small max size, it's necessary to:

1. use a good hash function that distributes keys evenly over the array indexes.
2. Sometimes resize the array.

... Though you <u>won't</u> do <u>all</u> this on the homework!