# ­Lab Assignment 27 (Hashing)

1. Write a program named **Lab27A** that will make a hash table of customer names and customer ID numbers. The customer ID number (integer) will be the key for each entry.
   1. Create a class named **Customer** that will have
      1. Two instance variables: the customer ID (int) and the customer name (String)
      2. A constructor that receives an int and a String parameter and fills in both instance variables.
      3. An int method to return the customer ID
      4. A toString method to print both var­iables (with labels as always)
   2. Back in the main class, do the following:
      1. Write an int method that receives an int parameter (the customer ID) and returns the result of performing the hash function on the parameter.

The hash function separates the key into digits and adds the digits together. If that sum has more than one digit, then the digits of the sum are added together. This will be repeated until the final result is only one digit long (hint: it must be < 10).

**Example**: hash(83945) = 8 + 3 + 9 + 4 + 5 = 29

But 29 has 2 digits so we do it again 🡪 2 + 9 = 11

11 still has 2 digits, so we do it one more time 🡪 1 + 1 = 2

Thus, hash(83045) = 2

* + 1. In the main method:
       1. Create an array of 10 **Customer** objects.
       2. Read the customer ID’s and names in from the text file (Lab27A.txt). (You don’t know how many customers there will be, but it will be < 10). For each one:
          1. Calculate the hash code by sending the customer ID to your hashing function method
          2. Write the customer’s name and hash function result
          3. Create a Customer object
          4. Add the object to your hash table array at the index returned by your hash function method. If there is a collision, then keep moving forward one array location at a time until you find an empty one (linear probing). Place the object there.

If you get to the end of the array, go back up to index 0 again.

* + - 1. After all customers have been added to the hash table, print the hash table array. (Don’t print the nulls, though.) Print the index number before each customer object’s info.  
         (You’ll probably need to write a for loop for this instead of using Arrays.toString.)

**Hint: To check if an array named arr has an object at location i, you would write:**

**if (arr[i] != null)**

1. Write a program named **Lab27B** where you will create a hash table of “gadgets” using a different hash function and quadratic probing.
   1. Create a class named **Gadget** that you will make objects of. It should have the following:
      1. Two instance variables: the gadget ID (int) and the gadget name (String)
      2. A constructor that receives an int and a String parameter and fills in both instance variables.
      3. An int method to return the gadget ID
      4. A toString method to print both variables (with labels)
   2. In your main class do the same things you did for Lab27A, but with following changes.
      1. The hash table array should have a length of 20.
      2. Read the data from Lab27B.txt
      3. The hash function will be **key % 20**
      4. You should use quadratic probing to find a new location for an entry if a collision occurs.
2. Program **Lab27C** will use an array of ArrayLists to simulate separate chaining in a hashing process.
   1. Write an int method that receives a string parameter. It should use the following hash function to calculate and return a hash value for the string:
      1. Look at the first letter of the string and set the hash value equal to the ordinal value for that letter (a = 1, b = 2,…z = 26). Covert the letter to lower case before doing this if needed.
      2. Add the length of the string to the hash value.
   2. In your main method
      1. Create an array of 36 ArrayLists. The ArrayLists will hold Strings.
      2. Read an unknown number of strings from a text file (Lab27C.txt) and call the above hash method for each string.   
         (I will test your program with a different file.)
      3. Use the returned hash value as the index in the array and add the string to that array position’s ArrayList.
      4. Print the array of ArrayLists. You should print each array index first for each “bucket” and then print the items in that ArrayList all on the same line. (If the ArrayList for that index is empty, then print the index and empty[].)