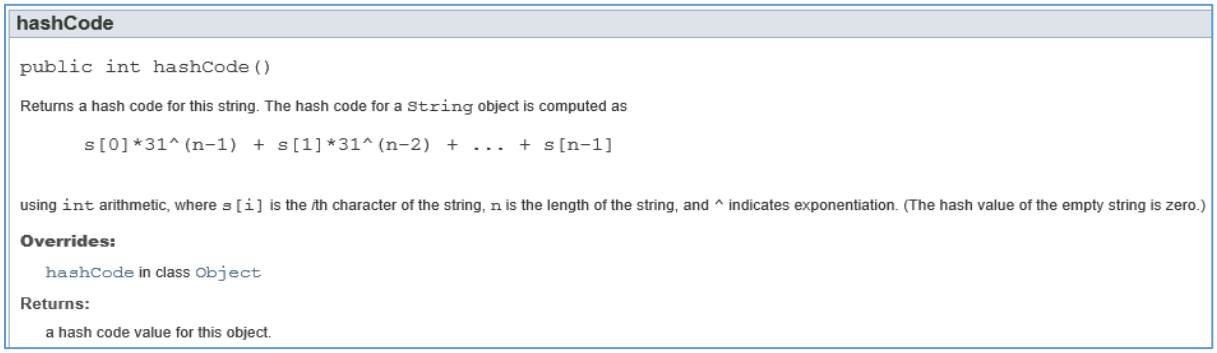
Lab 4: Hashing Strings

In this lab you will write a hashcode method that converts a String to an integer. Given a hashtable of size 1000000 (one million) and a file of over 300k words, you will calculate the hashcode of each string and find out many many of them result in collisions (under a separate chaining collision handler … so if two words happen to hash to the same index, only that one index is used up from the hash table.)

You will write a hash function: **`public static int hash(String s, int k)`**

which takes a String as input and computes the polynomial in k as seen in class. The following excerpt from official Java documentation indicates that Java uses a value k = 31 to computer this polynomial.



hashCode

`public int hashCode()`

Returns a hash code for this string. The hash code for a `String` object is computed as

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using `int` arithmetic, where `s[i]` is the *i*th character of the string, `n` is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

Overrides:

`hashCode` in class `Object`

Returns:

a hash code value for this object.

Your task is to simulate the hashing of all the English words in the given file using a polynomial for all values of k from 1 to 45. In each of these cases, you are to find the number of collisions that occurred, and also report the size of the largest hash bucket (the largest size of a chain under a separate chaining collision handler). Your hash function, on k=31, before you take it modulo 1million should result in the same number as Java's `.hashCode()` function, which is something you should check to see if you are computing the polynomials correctly.

Note that you do not have to implement a completely-functioning hashtable!

Firstly, you do not have to consider any deletions. Secondly, you do not have to store the words at each hash bucket, just the number of words that would be there.

Note also that this computed polynomial will overflow as an integer, causing it to wrap into negative values. You can decide on how to deal with it, but document it in your solution.

Your output should provide the requested information (your numbers might not match these exactly):

```
…k value = 31 resulted in 51789 collisions
k value = 31 maximum bucket size was 6
k value = 32 resulted in 220334 collisions
k value = 32 maximum bucket size was 1543…
```

(If you want to check your values against these values above, my hash function computed the polynomial value in an int variable, allowing it to overflow, then taking it modulo 1million, and if it resulted in a negative value, adding 1million to it.)

At the end of your code, in a block comment, report the 3 best k-values by measure of total number of collisions and the 3 bests k-value by having the best (i.e. smallest) maximum chain size.