Assignment 1 - Hash Functions

Problem 1:

HashF1 method explaination

hashF1 takes in a string

hashA int array defined on line 77

on line 83 we check if the string is less than 1 or more than 64 chars

and set ret to -1 if it is

if it isn't we add the string to our filler and cut it to 64 chars

we then loop through them adding to our hashA method

multiplying each numeric character value by prime numbers

line 97 to 100 we mod hashA to 255 for an 8bit ASCII

We then let ret equal to the hash values, with each element being

multiplied by 8bit for ASCII

this gives us our hash

Problem 2:

```
/*
 * Collision finder method
 */
private static void genCollision(int r) {
    Random random = new Random(); // create random instance

    // create string of all upper case, lower case, and numeric values
    String upp_Chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String chars = upp_Chars + upp_Chars.toLowerCase() + "0123456789";
    // String word will contain our randomized word
    String word = "";
    // tries variable to keep track of how many attempts it took to find collisions
    int tries = 0;
    // fixed int to limit word length
```

```
        final int MAX_WORD_LENGTH = 5;
        // fixed int to limit found words to 10;
        final int WORDS = 10;

        /*
         * For loop
         * to generate words that collide with hash function
         */
        for (int n = 0; n < WORDS; n++) {
            // continues loop until a word is found
            while (r != hashF1(word)) {
                word = ""; // reset word after each attempt
                for (int i = 0; i < MAX_WORD_LENGTH; i++) {
                    // randomly pick characters and numbers from chars string
                    word += chars.charAt(random.nextInt(chars.length() - 1));
                }
                tries++; // incremenet tries
            }

            // only runs when a word was found
            // prints out what the word was (input), that has equivalent (the same as orig
inal word), and the tries it took to find
            System.out.println("input = " + word + " : Hash = " + hashF1(word) + " : tries
= " + tries);
            // resets word
            word = "";
        }
    }
```

Problem 3:

```
/*
This method is to enchance our hash function
in the hashF1 method after we create our filler,
we reshuffle by calling
    filler = caesarShift(s, filler);
where s is our word (Bamb0) and filler is the filler we created
I also changed the filler to be less repetative making it less likely to result in the sam
e hash,
the new filler is now
    filler = new String("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890a
z");
*/

// Shifts the filler by the character numeric value of the key
    public static String caesarShift(String key, String filler) {
```

```
        int numericShift = 0;

        // for loop loops through the key (in this case Bamb0)
        for (int i = 0; i < key.length(); i++) {
            // gets the numeric character value of each character in the string and adds i
t to numericShift
            numericShift += Character.getNumericValue((char) key.charAt(i));
        }


        String newFiller = "";

        // runs through the filler length (in this case 64)
        for (int i = 0; i < filler.length(); i++) {
            // preform caesar shift
            // we find the shifted value of our filler string
            // and add it to our newFiller string
            newFiller += filler.charAt((numericShift + i) % (filler.length() - 1));
        }
        // return new filler string
        return newFiller;

    }
```