Dylan T Carlson

1161653

Lab110

11/08/18

**Client:**

/\*\*

\* This client class implements the hashCode cyclic-shift

\* computation described in the textbook. It includes the main method

\* calling the hashCode method on 3 similar strings.

\*

\* @author dylca

\*/

public class Client {

public static void main(String[] args) {

String a = "POTS";

String b = "STOP";

String c = "TOPS";

hashCode(a);

hashCode(b);

hashCode(c);

}

/\*\*

\* hashCode is passed a String and walks step by step

\* how it computes the hashCode using a cyclic-shift.

\*

\* @param s

\* @return

\*/

public static int hashCode(String s){

System.out.println("---------------------------------\n"+s);

int h = 0;

for(int i = 0; i < s.length(); i++){

System.out.print("\nEntering hashCode, pass " + i);

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString(h)).replace(" ", "0"));

System.out.print("\nhashCode <<5 ");

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString(h<<5)).replace(" ", "0"));

System.out.print("hashCode >>>27 ");

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString(h>>>27)).replace(" ", "0"));

h = (h << 5) | (h >>> 27); // 5-bit cyclic shift of the runing sum

System.out.print("hashCode <<5 | >>>27 ");

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString(h)).replace(" ", "0"));

h += (int) s.charAt(i); // add in next character

System.out.print("\nAdding character "+ s.charAt(i));

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString( (int) s.charAt(i) )).replace(" ", "0") );

System.out.print("Exiting hashCode");

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString(h)).replace(" ", "0"));

}

System.out.print("\nhash code for " +s+ " is ");

System.out.printf( " %32s \n", String.format("%32s", Integer.toBinaryString(h)).replace(" ", "0"));

return h;

}

}

**Output:**

run:

---------------------------------

POTS

Entering hashCode, pass 0 00000000000000000000000000000000

hashCode <<5 00000000000000000000000000000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000000000000000000000

Adding character P 00000000000000000000000001010000

Exiting hashCode 00000000000000000000000001010000

Entering hashCode, pass 1 00000000000000000000000001010000

hashCode <<5 00000000000000000000101000000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000000000101000000000

Adding character O 00000000000000000000000001001111

Exiting hashCode 00000000000000000000101001001111

Entering hashCode, pass 2 00000000000000000000101001001111

hashCode <<5 00000000000000010100100111100000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000010100100111100000

Adding character T 00000000000000000000000001010100

Exiting hashCode 00000000000000010100101000110100

Entering hashCode, pass 3 00000000000000010100101000110100

hashCode <<5 00000000001010010100011010000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000001010010100011010000000

Adding character S 00000000000000000000000001010011

Exiting hashCode 00000000001010010100011011010011

hash code for POTS is 00000000001010010100011011010011

---------------------------------

STOP

Entering hashCode, pass 0 00000000000000000000000000000000

hashCode <<5 00000000000000000000000000000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000000000000000000000

Adding character S 00000000000000000000000001010011

Exiting hashCode 00000000000000000000000001010011

Entering hashCode, pass 1 00000000000000000000000001010011

hashCode <<5 00000000000000000000101001100000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000000000101001100000

Adding character T 00000000000000000000000001010100

Exiting hashCode 00000000000000000000101010110100

Entering hashCode, pass 2 00000000000000000000101010110100

hashCode <<5 00000000000000010101011010000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000010101011010000000

Adding character O 00000000000000000000000001001111

Exiting hashCode 00000000000000010101011011001111

Entering hashCode, pass 3 00000000000000010101011011001111

hashCode <<5 00000000001010101101100111100000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000001010101101100111100000

Adding character P 00000000000000000000000001010000

Exiting hashCode 00000000001010101101101000110000

hash code for STOP is 00000000001010101101101000110000

---------------------------------

TOPS

Entering hashCode, pass 0 00000000000000000000000000000000

hashCode <<5 00000000000000000000000000000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000000000000000000000

Adding character T 00000000000000000000000001010100

Exiting hashCode 00000000000000000000000001010100

Entering hashCode, pass 1 00000000000000000000000001010100

hashCode <<5 00000000000000000000101010000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000000000101010000000

Adding character O 00000000000000000000000001001111

Exiting hashCode 00000000000000000000101011001111

Entering hashCode, pass 2 00000000000000000000101011001111

hashCode <<5 00000000000000010101100111100000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000000000010101100111100000

Adding character P 00000000000000000000000001010000

Exiting hashCode 00000000000000010101101000110000

Entering hashCode, pass 3 00000000000000010101101000110000

hashCode <<5 00000000001010110100011000000000

hashCode >>>27 00000000000000000000000000000000

hashCode <<5 | >>>27 00000000001010110100011000000000

Adding character S 00000000000000000000000001010011

Exiting hashCode 00000000001010110100011001010011

hash code for TOPS is 00000000001010110100011001010011

BUILD SUCCESSFUL (total time: 0 seconds)