***CT255 Assignment 1***

**Source Code for Problems 1 and 2.**

**Problem 1 Source Code:**

**\*\*To be inserted inside main()**

// Your code to look for a hash collision starts here!  
int foundColl = 0;  
int rndRes = 0;  
int totalSearches = 0;  
//Random number is generated a new object from the imported class  
Random randNum = new Random();  
String randomStr = "";  
//Continue looping until 10 collision strings are found  
while (foundColl<10) {  
 //Generate a random string of a random length (between 1 and the length of the input characters)  
 //This is done to generate similar strings of around the same length as the input string  
 randomStr = *randomString*((randNum.nextInt(args[0].length())+1));  
 rndRes = *hashF1*(randomStr);  
 //Total number of searches in incremented  
 totalSearches++;  
 //If the random string hash is the same as the input string hash, increase collision variable  
 //and print out the string and the resulting hash to the console  
 if(res == rndRes){  
 foundColl++;  
 System.*out*.println("\n\t\t\tCollision "+foundColl+" detected!\nString: "+randomStr+"\nHash: "+res);

res);}  
}  
//Total number of searches are displayed. This is done to compare with the new robust version   
//of hashF1  
System.*out*.println("\nIt took "+totalSearches+" searches to find 10 collisions!");  
//

\*\* **New method inside the class CT255\_HashFunction1 to generate a random string.**

static String randomString(int l){  
 //Ensures a 0 length string isn't passed into the function  
 if(l==0){l = 1;}  
 //List of characters and numbers to be used in the string  
 String stringChoice = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz0123456789";  
 int pos;  
 //A buffer is created for the desired size of the string  
 //Stringbuilder constructs a string with no characters in it with a specified length  
 StringBuilder rndStr = new StringBuilder(l);  
 //For loop to generate and append specific random characters to string  
 for ( int i=0; i<l; i++) {  
  
 //The length of the choice string along with math.random is used to generate a random position to choose  
 //from the string  
 pos = (int)(stringChoice.length() \* Math.*random*());  
 //The random character chosen is appended at the end of the random string.  
 rndStr.append(stringChoice.charAt(pos));  
 }  
 //At the end, random string is converted to a string and sent back to main.  
 return rndStr.toString();  
 }  
}

**Problem 2 Source Code:**

**\*\*Robust hashF1 method**

private static int hashF1(String s){  
 //More robust version of the hashF1 method. Goal is to ensure it takes significantly more searches to find collisions.  
 //i.e (harder to find collisions = lower risk of collision between input strings)  
 int ret = -1, i;  
 int[] hashA = new int[]{1, 1, 1, 1};  
   
 String filler, sIn;  
  
 //Replace 64 character buffer with character A-Z,a-z and 0-9, ABC is added to ensure its 64 characters.  
 //This will avoid repetition once the filler is concatenated with the input string  
 filler = new String("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz0123456789ABC");  
   
 if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length  
 ret = -1;  
 }  
 else {  
 sIn = s + filler;  
 sIn = sIn.substring(0, 64);  
 for (i = 0; i < sIn.length(); i++){  
 char byPos = sIn.charAt(i);  
 //The character selected will be multiplied by a constant times a particular character in the  
 //concatenated 64 character string. This means that the hash generated will be more depended  
 //on the particular string passed into the function.  
 hashA[0] += (byPos \* (17\*sIn.charAt(53))) ; // Note: A += B means A = A + B  
 hashA[1] += (byPos \* (31\*sIn.charAt(1)));  
 hashA[2] += (byPos \* (101\*sIn.charAt(4)));  
 hashA[3] += (byPos \* (79\*sIn.charAt(21)));  
 }  
 hashA[0] %= 255; // % is the modulus operation, i.e. division with rest  
 hashA[1] %= 255;  
 hashA[2] %= 255;  
 hashA[3] %= 255;  
  
 //Result  
 ret = hashA[0] + (hashA[1] \* 256) + (hashA[2] \* 256 \* 256) + (hashA[3] \* 256 \* 256 \* 256);  
 if (ret < 0) ret \*= -1;  
 }  
 return ret;  
 //------------Enhancements made------------  
 //Replaced filler with more characters to avoid repetition of characters.  
 //Inside the for loop, each element of the array is assigned the value of the characters at the i'th position  
 //times (A random constant times a random character anywhere in the string).  
 //With the original hashF1 method, it took 1729 attempts to find 10 collisions for the input string "Bamb0"  
 //With the robust version, it takes roughly around 15 million attempts to find 10 collisions for "Bamb0".  
}

**Entire Source Code for the File:**

*/\*\*  
 \*  
 \** ***@author*** *Michael Schukat  
 \*/*//Math and lang class are imported to use math.random function and stringbuilder class  
import java.util.\*;  
  
public class CT255\_HashFunction1 {  
  
 public static void main(String[] args) {  
 int res = 0;  
  
 if (args != null && args.length > 0) { // Check for <input> value  
 res = *hashF1*(args[0]); // call hash function with <input>  
 if (res < 0) { // Error  
 System.*out*.println("Error: <input> must be 1 to 64 characters long.");  
 }  
  
 else {  
 System.*out*.println("input = " + args[0] + " : Hash = " + res);  
 System.*out*.println("------Start searching for collisions------");  
 // Your code to look for a hash collision starts here!  
 int foundColl = 0;  
 int rndRes = 0;  
 int totalSearches = 0;  
 //Random number is generated a new object from the imported class  
 Random randNum = new Random();  
 String randomStr = "";  
 //Continue looping until 10 collision strings are found  
 while (foundColl<10) {  
 //Generate a random string of a random length (between 1 and the length of the input characters)  
 //This is done to generate similar strings of around the same length as the input string  
 randomStr = *randomString*((randNum.nextInt(args[0].length())+1));  
 rndRes = *hashF1*(randomStr);  
 //Total number of searches in incremented  
 totalSearches++;  
 //If the random string hash is the same as the input string hash, increase collision variable  
 //and print out the string and the resulting hash to the console  
 if(res == rndRes){  
 foundColl++;  
 System.*out*.println("\n\t\t\tCollision "+foundColl+" detected!\nString: "+randomStr+"\nHash: "+res);  
 }  
 }  
 //Total number of searches are displayed. This is done to compare with the new robust version  
 //of hashF1  
 System.*out*.println("\nIt took "+totalSearches+" searches to find 10 collisions!");  
 //  
 }  
 }  
 else { // No <input>   
 System.*out*.println("Use: CT255\_HashFunction1 <Input>");  
 }   
 }  
   
 private static int hashF1(String s){  
 //More robust version of the hashF1 method. Goal is to ensure it takes significantly more searches to find collisions.  
 //i.e (harder to find collisions = lower risk of collision between input strings)  
 int ret = -1, i;  
 int[] hashA = new int[]{1, 1, 1, 1};  
   
 String filler, sIn;  
  
 //Replace 64 character buffer with character A-Z,a-z and 0-9, ABC is added to ensure its 64 characters.  
 //This will avoid repetition once the filler is concatenated with the input string  
 filler = new String("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz0123456789ABC");  
   
 if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length  
 ret = -1;  
 }  
 else {  
 sIn = s + filler;  
 sIn = sIn.substring(0, 64);  
 for (i = 0; i < sIn.length(); i++){  
 char byPos = sIn.charAt(i);  
 //The character selected will be multiplied by a constant times a particular character in the  
 //concatenated 64 character string. This means that the hash generated will be more depended  
 //on the particular string passed into the function.  
 hashA[0] += (byPos \* (17\*sIn.charAt(53))) ; // Note: A += B means A = A + B  
 hashA[1] += (byPos \* (31\*sIn.charAt(1)));  
 hashA[2] += (byPos \* (101\*sIn.charAt(4)));  
 hashA[3] += (byPos \* (79\*sIn.charAt(21)));  
 }  
 hashA[0] %= 255; // % is the modulus operation, i.e. division with rest  
 hashA[1] %= 255;  
 hashA[2] %= 255;  
 hashA[3] %= 255;  
  
 //Result  
 ret = hashA[0] + (hashA[1] \* 256) + (hashA[2] \* 256 \* 256) + (hashA[3] \* 256 \* 256 \* 256);  
 if (ret < 0) ret \*= -1;  
 }  
 return ret;  
 //------------Enhancements made------------  
 //Replaced filler with more characters to avoid repetition of characters.  
 //Inside the for loop, each element of the array is assigned the value of the characters at the i'th position  
 //times (A random constant times a random character anywhere in the string).  
 //With the original hashF1 method, it took 1729 attempts to find 10 collisions for the input string "Bamb0"  
 //With the robust version, it takes roughly around 15 million attempts to find 10 collisions for "Bamb0".  
 }  
  
 static String randomString(int l)  
  
 {  
 //Ensures a 0 length string isn't passed into the function  
 if(l==0){  
 l = 1;  
 }  
 //List of characters and numbers to be used in the string  
 String stringChoice = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvxyz0123456789";  
 int pos;  
  
 //A buffer is created for the desired size of the string  
 //Stringbuilder constructs a string with no characters in it with a specified length  
 StringBuilder rndStr = new StringBuilder(l);  
  
 //For loop to generate and append specific random characters to string  
 for ( int i=0; i<l; i++) {  
  
 //The length of the choice string along with math.random is used to generate a random position to choose  
 //from the string  
 pos = (int)(stringChoice.length() \* Math.*random*());  
  
 //The random character chosen is appended at the end of the random string.  
 rndStr.append(stringChoice.charAt(pos));  
 }  
 //At the end, random string is converted to a string and sent back to main.  
 return rndStr.toString();  
 }  
  
}