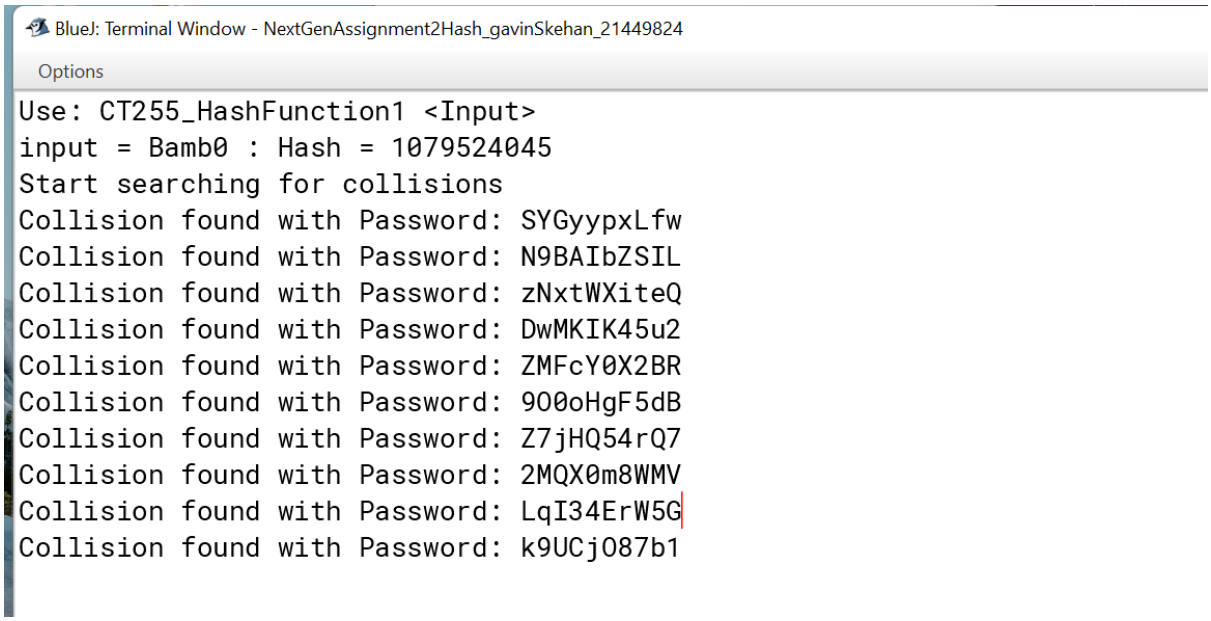
Next-Generation Technologies Assignment 1 – Hash functions

Code summary

- We create our hash function class where we initialise our res value which acts as our hash value.
- We use the if else statements to determine the input value, ensuring that we are given an appropriate value (must be 1-64 characters). If the input is valid we print out the hash code and the code looks for the hash collision else there's no input.

- The input is sent to the hash method, which determines whether it is a valid input.
- Once the has function determines that it is legitimate, filler is concatenated onto s before sIn is cut to 64 characters.
- The for loop in line 42 selects one of the four points in the hashA array, multiplies the corresponding position in the string by a random integer, and then adds the result to the array's number.
- The array's numbers are then each multiplied by 255.
- Each integer is added together, multiplied by 256, and then divided by the array's position.

Problem 2



Problem 2 code:

```
import java.security.SecureRandom;

/**

 *
```

```
 * @author Gavin Skehan
 */
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!

            // characters the password generator can use
            String ALPHA_CAPS="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            String ALPHA="abcdefghijklmnopqrstuvwxyz";
            String NUMERIC = "0123456789";

            int i = 0;

            while(i<10){
               String password = randomPassword(10, ALPHA_CAPS + ALPHA + NUMERIC);
               // Call password with 10 characters long with the above character variables
               int tempHash = hashF1(password); // hashing the password so we can comapre
               // it to the hash of the input
               if(tempHash==res) { // check to see if the hash is the same
```

```
                System.out.printf("Collision found with Password: %s\n", password);
                // print collision
                i++;
              }

            }
    }
  }
    else { // No <input>
    System.out.println("Use: CT255_HashFunction1 <Input>");
  }
}

  private static int hashF1(String s){
    int ret = -1, i;
    int[] hashA = new int[]{1, 1, 1, 1};

    String filler, sIn;

    filler = new String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGH");

    if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length
      ret = -1;
    }
    else {
      sIn = s + filler; // Add the characters
      sIn = sIn.substring(0, 64); // // Limits the string to first 64 characters
      for (i = 0; i < sIn.length(); i++){
```

```
            char byPos = sIn.charAt(i); // get i'th character
            hashA[0] += (byPos * 17);
            hashA[1] += (byPos * 31);
            hashA[2] += (byPos * 101);
            hashA[3] += (byPos * 79);
        }

        hashA[0] %= 255;  // % is division with remainder (modulus)
        hashA[1] %= 255;
        hashA[2] %= 255;
        hashA[3] %= 255;

        ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 *
256);
        if (ret < 0) ret *= -1;
      }
      return ret;
   }

   private static String randomPassword(int len, String dic){ // random password generator
      SecureRandom random = new SecureRandom();
      String result = "";
      for(int i = 0;i < len;i++){
         int index = random.nextInt(dic.length()); // pick random number between the range
         result += dic.charAt(index); // Adds character at position of index
      }
      return result; // returns result to string password
   }
}
```

Problem 3

A collision occurs when more than one value to be hashed by a particular hash function, hash to the same slot in the table or data structure (hash table) being generated by the hash function. We randomised the indexes which results in less collisions as the values hash to different slots in the table or data structure. i.e i % 4…..

Options

```
Use: CT255_HashFunction1 <Input>
input = Bamb0 : Hash = 110064280
Start searching for collisions
```

```
import java.security.SecureRandom;
/**
 *
 * @author Gavin Skehan
 */
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!
```

```
            // characters the password generator can use
            String ALPHA_CAPS="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            String ALPHA="abcdefghijklmnopqrstuvwxyz";
            String NUMERIC = "0123456789";

            int i = 0;

            while(i<10){
               String password = randomPassword(10, ALPHA_CAPS + ALPHA + NUMERIC);
               // Call password with 10 characters long with the above character variables
               int tempHash = hashF1(password); // hashing the password so we can comapre
               // it to the hash of the input
               if(tempHash==res) { // check to see if the hash is the same
                  System.out.printf("Collision found with Password: %s\n", password);
                  // print collision
                  i++;
               }

            }
      }
   }
      else { // No <input>
      System.out.println("Use: CT255_HashFunction1 <Input>");
   }
}

   private static int hashF1(String s){
      int ret = -1, i;
```

```
    int[] hashA = new int[]{1, 1, 1, 1};


    String filler, sIn;


    filler = new
String("ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEF
GH");


    if ((s.length() > 64) || (s.length() < 1)) { // String does not have required length
       ret = -1;
    }
    else {
       sIn = s + filler; // Add characters
       sIn = sIn.substring(0, 64); // // Limit string to first 64 characters
       for (i = 0; i < sIn.length(); i++){
          char byPos = sIn.charAt(i); // get i'th character
          hashA[i % 4] += (byPos * 17); // randomise so there are less collisions
          hashA[(i+1) % 4] += (byPos * 31);
          hashA[(i+2) % 4] += (byPos * 101);
          hashA[(i+3) % 4] += (byPos * 79);
       }


       hashA[0] %= 255;  // % is division with remainder (modulus)
       hashA[1] %= 255;
       hashA[2] %= 255;
       hashA[3] %= 255;


       ret = hashA[0] + (hashA[1] * 256) + (hashA[2] * 256 * 256) + (hashA[3] * 256 * 256 *
256);
       if (ret < 0) ret *= -1;
```

```
    }
    return ret;
  }

  private static String randomPassword(int len, String dic){ // random password generator
    SecureRandom random = new SecureRandom();

    String result = "";
    for(int i = 0;i < len;i++){ // repeats 10 times
      int index = random.nextInt(dic.length()); // pick random number between the range
      result += dic.charAt(index); // Adds character at position of index
    }
    return result; // returns result to string password
  }
}
```