	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 7

No de Práctica(s): 3

Integrante(s): 322044339

322094152

322114461

425093384

322150984

No. de brigada: 6

Semestre: 2026-1

Fecha de entrega: 10 de septiembre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

Frecuentemente, cuando programamos, debemos pensar cómo guardar o representar la información de una forma segura y que no ocupe mucha memoria. Una de esas formas es usando hash o "funciones digestivas". Imaginemos que se trata de una licuadora, en donde metes algo y lo que tienes como resultado es un licuado de lo que introdujiste. Las funciones digestivas devuelven una salida donde tiene mezcladas letras y números a partir de una entrada, como es una cadena. Suena sencillo, pero si no se hace bien puede devolver la misma salida siempre, volviéndose tan predecibles que pierde su gracia. [1]

Actualmente, con tanta información privada dando vueltas por internet, desde contraseñas hasta datos personales, usar hashes débiles puede ser riesgoso, pues cualquiera con conocimientos en el tema y malas intenciones puede revertirlos y saber su contenido. Por eso es sumamente importante entenderlos, aunque sea algo simplificado para esta práctica.

Conocer la manera de aplicar esto, a pesar de ser solo una simulación de hash, nos es útil para dar un paso al entendimiento en dirección a la criptografía. También funciona para practicar el manejo de estructuras de datos, como son listas y diccionarios, empleados para organizar las entradas y sus hashes, y entender la importancia de la pseudoaleatoriedad en la programación. [5]

Para esta práctica se desarrollará un programa que simule el comportamiento de una función digestiva, la cual tomará varias cadenas de texto desde la terminal y generará para cada una un código hexadecimal de 32 caracteres asociado, que representará la encriptación de la cadena. Posteriormente, se imprime la cadena ingresada junto a su versión encriptada. Se busca tener registro de tanto la solución teórica como su implementación en la práctica, aplicando ArrayList y HashMap con el propósito de entender su funcionamiento, así como validar el correcto desempeño del programa y de la función digestiva mediante pruebas con diferentes entradas y el análisis de los resultados obtenidos.

2. Marco Teórico

- Array List

  La clase Array List permite el uso de todas las operaciones con listas conocidas y permite introducir todo tipo de elementos, incluido null, además, esta clase permite trabajar con el tamaño de la matriz en la que se almacenan los datos internamente mediante el uso de métodos.

  Cada ArrayList tiene una capacidad relacionada con el tamaño de la matriz, esto quiere decir que la matriz es "dinámica"pues mientras se agregan elementos al ArrayList, aumenta su capacidad y por ende el tamaño de la matriz.

Hablando de gestión del tiempo, las operaciones "size", ïsEmpty", "get", "set", ïteratorz "listIterator"son operaciones de tiempo constante, mientras que .ªdd.es de orden n.[2]

- Hash Table

  Es una clase que implementa una tabla que realiza un mapeo de claves con valores, relacionándolos uno a uno, cualquier elemento no null, puede ser utilizado como clave o valor.

  Para el almacenamiento y recuperación de elementos dentro del HashTable, resulte exitoso, los elementos utilizados como clave, deberán ser implementados mediante los métodos equalsz "hashCode".

  Un Hash Table consta de dos parámetros que afectarán su rendimiento: capacidad inicial y factor de carga; la capacidad inicial se refiere a la cantidad de "buckets"(contenedores que almacenan uno o más pares de clave-valor) con los que cuenta el Hash Table al momento de su creación, mientras que el factor de carga es una medida de qué tan "lleno"puede estar el Hash Table antes de aumentar su capacidad automáticamente, ambos elementos son muy importantes pues, la capacidad inicial controla la compensación entre el espacio desperdiciado y la necesidad de realizar operaciones para aumentar el tamaño que consumen demasiado tiempo, mientras que el factor de carga (que por defecto es de 0.75) es una compensación entre el tiempo y la sobrecarga de espacio (datos que se utilizan para la gestión de los datos con los que el programa trabaja)[4]

- Hash Map

  Su equivalente más cercano es el Hash Table, con la diferencia de que sí permite elementos "null"; la implementación de un Hash Map garantiza un rendimiento constante en el uso de operaciones "putz "get.ªdemás que dispersa sus elementos de forma adecuada entre sus "buckets".

  Además, de igual manera que con Hash Table, cuenta con dos parámetros que afectan su rendimiento: capacidad inicial y factor de carga.[3]

3. Desarrollo

Para este código tuvimos como objetivo el generar mediante una función digestiva cadenas pseudoaleatorias que recibiera cadenas de texto y las transformara a un formato hexadecimal, por lo que para facilitar las cosas y tomando la plantilla elaborada en la sesión teórica decidimos modificarla para poder lograr el objetivo.

Primero importamos tres bibliotecas siendo java.util.ArrayList, java.util.HashMap y java.util.Random las cuales utilizaremos para instanciar objetos de esas clase.

Los dos métodos principales del código son el método main y el método generarHash, este último está encargado de la función digestiva por lo que como parámetro recibe una cadena y devuelve otra. Dentro de este se inicializa una semilla en 0 que en el futuro servirá para la generación de números aleatorios, después con "texto.toCharArray()"la cadena ingresada se volverá un arreglo de caracteres y con un for each se recorrerá la cadena, en cada iteración la semilla va recolectando los caracteres. Después se crea un objeto random inicializado con la semilla y un objeto StringBuilder para formar la cadena final a través de un bucle for con 32 iteraciones generando un Hash de 32 caracteres, una variable valor que con random genera números aleatorios desde 0 a 15 y con append le añadimos cada número en forma hexadecimal a la StringBuilder, al final de la función digestiva devolviendo la cadena generada.

Ya en la función main que es la parte que nos tocaba diseñar se tiene que agregar código para que todo funciones y los objetivos se cumplan, primero definimos a un objetivo ArrayList de nombre list que almacenará cadenas y un objeto HashMap de nombre map en la que tanto su llave como valor son cadenas. Con un el primer for se recorre args y los argumentos se agregan a list, con el segundo for se generan los hash que almacenamos en una variable de nombre code, por lo que se llama al método generarHash enviándole elementos del ArrayList para que los transforme a las cadenas hexadecimales que buscábamos, ya teniendo la llave y el valor, simplemente los incluimos en el mapa con ".put(list.get(i), code" por lo que ya cada llave está asociada a una entrada. Después inicializamos dos cadenas vacías, una "inp" y "out", la primera almacenará la frase originalmente ingresada y la otra contendrá los hash, y mediante un for se recorre la lista y se van construyendo las cadenas procurando un orden similar en su impresión.

Por lo tanto y ya explicando el código podemos decir que si ingresamos una cadena de texto está separará las palabras y para cada una el programa generará una cadena encriptada en hexadecimal aunque para aplicaciones realistas no es muy segura. Entonces, supongamos que como argumentos le mandamos la cadena "Hola Mundo" entonces se esperaría que en ArrayList se almacenara "Hola", "Mundo", en la generación del Hash con un objeto random dado la semilla como parámetro además de un Integer.toHexString se generasen dos cadenas de tipo hexadecimal tales como las que se presentarán más adelante, esto se almacenaría en un HashMap y se imprimiría como se muestra a continuación.

Lo mismo sucede si ingresamos la oración "Facultad de Ingeniería", cada elemento de la cadena se almacenará en ArrayList y gracias a Hash, se genera su encriptación.

4. Resultados

4.1. Hola Mundo

```
PS C:\Users\OEM\OneDrive\Facultad de Ingeniería\Tercer Semestre\POO> java Practica3 Hola Mundo
La encriptación relacionada a tu frase es:
Frase: Hola Mundo
Encriptación: c9d3ef43eaa48e1e0a8fde0a176039ac a4ef0e40c9220e0eabc2b7dc7ef06ddf
PS C:\Users\OEM\OneDrive\Facultad de Ingeniería\Tercer Semestre\POO>
```

Figura 1: Encriptación de la cadena de texto ”Hola Mundo”

Se ejecuta el programa y se le da ahí mismo una cadena de texto, luego de que la función digestiva hace su trabajo, se muestra la encriptación de las palabras de la cadena.

4.2. Facultad de Ingeniería

```
La encriptación relacionada a tu frase es:
Frase: Facultad de Ingeniería
Encriptación: b6b82af5b1d043dcb4b0fdf73820cc58 b0e845853a5c98792321841d03783da7 ac4d81c46d9b60debf4187eb8dd53fb5
```

Figura 2: Encriptación de la cadena de texto ”Facultad de Ingeniería”

Se ejecuta el programa por segunda vez pero ahora la cadena de texto cambia, luego de que la función digestiva hace su trabajo, se muestra la encriptación de las palabras de esta nueva cadena.

5. Conclusiones

Esta práctica permitió ver que los conceptos de funciones digestivas y el uso de estructuras en Java (como HashMap, ArrayList, etc.) se utilizan directamente para manipular objetos del tipo cadena de texto. En particular, se le dio sentido al arreglo args del método main, el cual recibe los argumentos enviados por consola y los transforma en valores pseudoaleatorios en formato hexadecimal mediante una función digestiva que usa una semilla como punto de partida.

Este desarrollo nos permitió comprender cómo se procesan este tipo de datos de entrada, cómo se almacenan en çolecciones"dinámicas y de qué manera se pueden asociar claves y valores en un mapa para organizar la información. A diferencia de C, donde estas estructuras deben implementarse manualmente mediante struct, en Java ya se encuentran listas para usarse, lo cual facilita la resolución de problemas de este tipo y abre más posibilidades para enfocarse en la lógica del programa.

6. Referencias bibliográficas

Referencias

[1] Alexander Obregón. *What are Hashcodes in Java? A Simple Guide*. Abr. de 2024. URL: https://medium.com/@AlexanderObregon/what-is-hashcode-in-java-a-simple-guide-a3b95d6ebae4 (visitado 07-09-2025).

[2] Oracle. *Class ArrayList<E>, Java Platform, Standard Edition 8 API Specification*. URL: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html (visitado 09-09-2025).

[3] Oracle. *Class HashMap<K, V>, Java Platform, Standard Edition 8 API Specification*. URL: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html (visitado 08-09-2025).

[4] Oracle. *Class Hashtable<K, V>, Java Platform, Standard Edition 8 API Specification*. URL: https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html (visitado 08-09-2025).

[5] César Soto Valero. *Encoding, Encryption, Hashing, and Obfuscation in Java*. Dic. de 2021. URL: https://www.cesarsotovalero.net/blog/encoding-encryption-hashing-and-obfuscation-in-java.html (visitado 07-09-2025).