	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: 1

No de Práctica(s): 3

Integrante(s): 322184022

322238682

321570789

322069093

322249723

No. de lista o brigada: 5

Semestre: 2026-1

Fecha de entrega: 12/09/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

En esta práctica, se trabajó con el lenguaje de programación Java y se utilizaron estructuras de datos como ArrayList y HashMap. La actividad consistió en introducir cadenas de texto, almacenarlas en un arreglo y generarles un valor hash. Se utilizó HashMap para darle a cada cadena su respectivo hash. Aplicar esto sirve para entender cómo se generan los identificadores únicos, aunque cabe recalcar que son pseudoaleatorios, es decir, no tan unicos, pero útiles para el manejo de la información, el funcionamiento de los hash, los cuales sirven para la búsqueda de datos y la verificación de su integridad, así como la asignación de ciertas claves a los datos con los que se trabaja.

Objetivos

- Comprender las estructuras de datos ArrayList y HashMap en Java para introducir cadenas de texto y crearles valores Hash.
- Aprender cómo se generan valores hash a partir de cadenas de texto.
- Aplicar lo aprendido sobre la generación y el almacenamiento de hash.

2. Marco Teórico

En esta práctica, se aplica el concepto visto en clase de función digestiva o *función Hash*, que se refiere a un algoritmo matemático que toma cualquier conjunto de datos de entrada (como un archivo, texto, imagen, etc.) y lo transforma en una cadena alfanumérica de longitud fija, conocida como valor hash. [4]
La característica distintiva de una función 'hash' es que, partiendo de un determinado conjunto de datos de entrada, siempre se generará el mismo valor alfanumérico. Es decir, es un código único para esos datos. [1]
Igualmente, en clase se vio lo que es el *determinismo computacional*, que quiere decir que una entrada específica siempre produce la misma salida cuando se procesa con el mismo algoritmo. A pesar de esto, se puede simular aleatoriedad mediante generadores de números pseudoaleatorios con una *semilla* para producir secuencias que parecen aleatorias pero son reproducibles con la misma semilla. [3]

3. Desarrollo

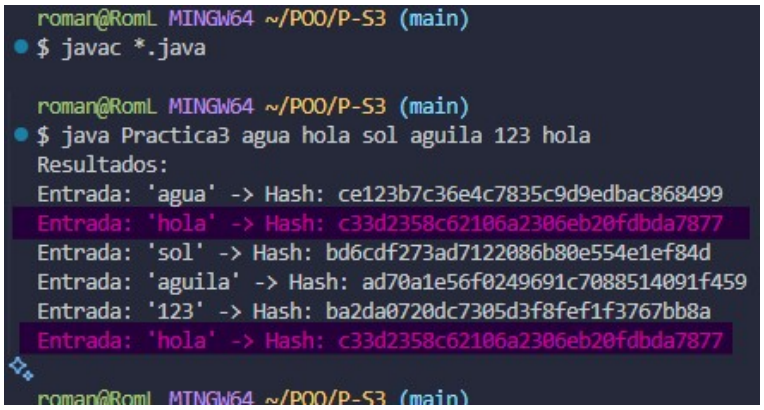
Usamos distintas clases, como *ArrayList*, la cual sirve para almacenar dinámicamente las entradas recibidas desde la línea de comandos, *HashMap*, guarda cada entrada junto con el hash generado y *Random* produce números aleatorios que forman la base del hash.

StringBuilder
Dado que la clase String en JAVA no permite modificaciones, implementamos *StringBuilder* esta clase permite hacer cambios directamente dentro de la secuencia de caracteres.
Sabemos que al crear un objeto en String para poder modificar su valor será necesario crear mas objetos y guardar sus nuevos valores, esto nos puede generar residuos, por lo que al utilizar *StringBuilder* podríamos modificar nuestra secuencia de carácteres sin tener que crear un string nuevo, reduciendo los residuos y mejorando el rendimiento. [2]

- **StringBuilder()**: Genera un StringBuilder vacío con una capacidad máxima de 16 caracteres.
- **append()**: Se usa para añadir un string a otro.
- **semilla**: Es el valor inicial utilizado para inicializar un generador de números pseudoaleatorios, se utiliza para inicializar el objeto Random (se lo da como argumento) para asegurar que el al generar el numero aleatorio, produzca la misma secuencia cada vez que se inicializa con la misma semilla.
- **Inter.toHexString**: es un método estático de la clase *Integer*, nos ayudó a convertir los números aleatorios que se generaron en su equivalente hexadecimal para formar el código hash.

4. Resultados

Ejecución:



La salida muestra los valores hash generados para cada cadena dada al método main (agua, hola, sol, aguila, 123, hola).

Podemos destacar que:

- Para cada valor *diferente* se generó un Hash *diferente*, aleatoriamente
- Para valores *iguales* de generaron los mismos hash, ya que al inicializar Random con la *semilla*, se genera la misma secuencia de caracteres en todas las llamadas iguales, por lo tanto, en este caso, cada vez que procesa "hola"genera el mismo hash. Aquí podemos ver cómo se cumple el concepto del determinismo explicado en el *Marco teórico*, que la misma entrada siempre produce la misma salida, ya que tienen la misma ***semilla***.

5. Conclusiones

La práctica permitió aplicar los conceptos teóricos de funciones hash y determinismo computacional vistos en la clase. El código simula una función digestiva que produce hashes iguales para entradas iguales, como se pudo ver con la repetición del hash para ”hola” en la sección de *Resultados*. El uso de ArrayList y HashMap nos ayudó el manejo de los datos, y la generación pseudoaleatoria con Random hizo que los hashes fueran únicos para entradas diferentes.

6. Referencias

[1] BBVA. De la verificación de contraseña a la firma electrónica: el secreto está en el 'hash'. (https://www.bbva.com/es/innovacion/de-la-verificacion-de-contrasena-a-la-firma-electronica-el-secreto-esta-en-el-hash/), marzo 27, 2025. Consultado: 10-09-2025.

[2] IONOS. Java stringbuilder: funcionamiento de la clase modificable. (https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/java-stringbuilder/), enero, 2025. Consultado: 10-09-2025.

[3] J. Rodríguez. Generadores de números aleatorios y su aplicación para cifrar y autenticar datos digitales. (https://share.google/3HbopMGYariApITTh), noviembre, 2024. Consultado: 10-09-2025.

[4] Signaturit. ¿qué es un hash y cómo funciona? (https://www.signaturit.com/es/blog/que-es-un-hash/), marzo 24, 2025. Consultado: 10-09-2025.