	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Dávila Pérez René Adrián

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): 4

Integrante(s): 322089020

322089817

322151194

425091586

322085390

No. de lista o brigada: Equipo 4

Semestre: 2026-1

Fecha de entrega: 12 de septiembre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

Planteamiento del Problema

El ejercicio consiste en desarrollar un programa que reciba una cadena desde el método main y, basándose en el concepto de una función digestiva, producir una salida que simule una función Hash, este problema requiere la aplicación de los conocimientos teóricos vistos en clase para crear una solución funcional.

Motivación

Para llevar a cabo esta práctica es necesario traducir el conocimiento teórico en una solución práctica, al implementar la simulación de una función digestiva, se refuerzan los conceptos de programación orientada a objetos, demostrando su relevancia y aplicación directa en la resolución de problemas computacionales.

Objetivos

El objetivo principal de la práctica es implementar la solución teórica del ejercicio visto en clase para crear una aplicación que simule una función digestiva.

- Desarrollar la aplicación en el lenguaje JAVA aplicando los conceptos vistos en clase.
- Asegurar que la aplicación tome la cadena de texto desde el método main.
- Generar una cadena pseudoaleatorio con base a una semilla dada, como resultado de la simulación de una función digestiva Hash.

2. Marco Teórico

Función Digestiva (Hash)

Una función digestiva o hash es un algoritmo que toma una entrada de longitud variable y la transforma en una salida de longitud fija, generalmente representada como una cadena de caracteres pseudoaleatoria. Esta transformación es determinista, lo que significa que la misma entrada siempre produce la misma salida. Las funciones hash son fundamentales en criptografía, ya que permiten verificar la integridad de los datos y autenticar la información sin necesidad de revelar el contenido original. [2]

Semilla

La semilla es un valor inicial que se utiliza para iniciar un generador de números pseudoaleatorios. En esta práctica, la semilla se obtiene sumando los valores ASCII de los caracteres de la cadena de entrada. Este enfoque garantiza que entradas diferentes produzcan resultados distintos, mientras que la misma entrada siempre genera la misma salida. La clase Random de Java utiliza una semilla para generar secuencias de números pseudoaleatorios deterministas. [3]

Números Pseudoaleatorios

Un generador de números pseudoaleatorios (PRNG) es un algoritmo que produce una secuencia de números que simulan la aleatoriedad, pero que son deterministas si se conoce la semilla inicial. En Java, la clase Random implementa un PRNG que utiliza una semilla de 48 bits y un algoritmo congruencial lineal para generar números pseudoaleatorios. [3]

Colecciones en Java

En Java, las colecciones son estructuras de datos que permiten almacenar y manipular grupos de objetos. La clase ArrayList implementa la interfaz List y permite almacenar elementos en una secuencia ordenada, mientras que la clase HashMap implementa la interfaz Map y permite almacenar pares clave-valor, proporcionando una forma eficiente de asociar claves con valores. Estas colecciones son fundamentales para organizar y manejar datos de manera eficiente en programas Java. [1]

3. Desarrollo

Implementación del Código

Para la ejecución del programa no se hace uso de la clase Scanner, ya que las entradas no se solicitan al usuario durante la ejecución, sino que se reciben directamente como parámetros en el método main. Esto significa que al momento de ejecutar el programa en consola, el usuario debe escribir las palabras que se desean procesar. Cada una de estas palabras es tratada como un argumento independiente.

El programa utiliza dos estructuras principales de la biblioteca de Java: un ArrayList y un HashMap. El ArrayList tiene la función de almacenar de manera ordenada todas las palabras ingresadas, sin importar cuántas sean, ya que este tipo de estructura puede crecer dinámicamente. Por otro lado, el HashMap sirve para relacionar cada palabra con un valor generado por el programa, que en este caso es un hash simulado en formato hexadecimal.

El funcionamiento general se puede resumir en los siguientes pasos: primero, las palabras se guardan en el ArrayList. Después, por cada palabra, se aplica un método llamado generaHash, que produce una cadena de 32 caracteres hexadecimales asociada de manera única a esa palabra. Finalmente, cada palabra junto con su hash se almacena en el HashMap y posteriormente se muestran los resultados en pantalla. Con esta estrategia, el programa puede procesar múltiples entradas de forma ordenada y mostrar de manera clara las relaciones entre cada entrada y su respectiva salida.

Funciones

generaHash: Esta función recibe como parámetro un texto y devuelve una cadena de 32 caracteres hexadecimales que representan un hash. El procedimiento consiste primero en calcular una semilla a partir de la suma de los valores numéricos de todos los caracteres que componen el texto. Dicha semilla se utiliza para inicializar un objeto tipo Random, lo que garantiza que para la misma palabra siempre se genere el mismo hash pseudoaleatorio.

Una vez inicializado el generador aleatorio, se procede a construir el hash. Para ello se repite un ciclo 32 veces, y en cada iteración se obtiene un número entero entre 0 y 15, el cual se transforma a su equivalente hexadecimal (un dígito entre 0–9 o a–f). La concatenación de estos valores da como resultado una cadena de 32 caracteres, que se devuelve como salida de la función. De esta manera, el método asegura que cada palabra tenga un valor identificador único, de manera similar al funcionamiento de los algoritmos de encriptación, aunque en este caso se trata de una simulación.

main: El método principal comienza validando si el usuario ha proporcionado argumentos al ejecutar el programa. En caso contrario, se imprime un mensaje indicando que es necesario introducir al menos una palabra. Si se cuenta con entradas, estas se agregan a un ArrayList para mantenerlas organizadas. Luego, el programa crea un HashMap donde la clave será la palabra original y el valor será el hash correspondiente, obtenido al invocar la función generaHash. Una vez que todas las palabras han sido procesadas, el programa recorre nuevamente el ArrayList para imprimir cada palabra con su hash correspondiente.

Pruebas

```
Input(Args): Un Día vi Una Vaca Sin Cola Vestida De Uniforme
Output: Resultados:
    Entrada: Un-> Hash:b53890be869cc1ed743f53ee889e8893
    Entrada: Día-> Hash:ca196f14c3d7de1581912b33ddc0bee5
    Entrada: vi-> Hash:b485429b0ae27e192c3bdd0cfea74652
    Entrada: Una-> Hash:caf5257e254415e4ebdb6104bab156a1
    Entrada: Vaca-> Hash:b59346580b9293976fa99b2a385256b1
    Entrada: Sin-> Hash:c24f5c0fe3a047c707d5a47d0fad7731
    Entrada: Cola-> Hash:b746857b874ce66ba8ad120cd24998f3
    Entrada: Vestida-> Hash:b07a077383004b8c49291f93d7cd7f26
    Entrada: De-> Hash:b67e007f6fcf01e4fc41a0185ffe0edc
```

```
    Entrada: Uniforme-> Hash:b45ffac3ec0b635af994fa28171884e3
```

```
Input(Args):
```

```
Output:
```

```
    Ingresa una palabra al menos
```

4. Resultados

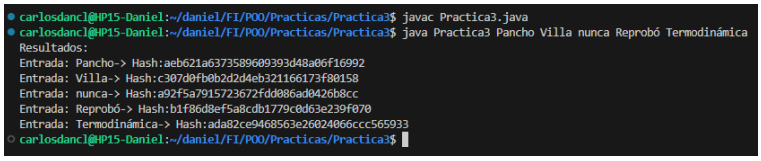

Figura 1: Se ejecuta el programa con una cadena cómo argumento, imprime el resultado

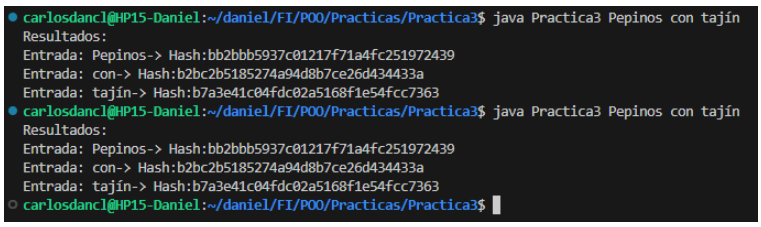

Figura 2: Se ingresa la misma cadena cómo argumento mostrando el mismo resultado

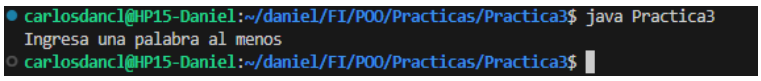

Figura 3: Se ejecuta el programa sin ingresar un argumento

5. Conclusiones

El desarrollo e implementación de esta práctica permitió aplicar de manera concreta los conceptos teóricos abordados en clase, cumpliendo con los objetivos planteados. Asimismo, se logró simular una función digestiva, demostrando la capacidad de la programación orientada a objetos para resolver problemas complejos de manera estructurada.

A través de este ejercicio, se reafirmo la importancia de la aplicación de los conocimientos teóricos para la creación de soluciones funcionales, el proceso de generar una cadena pseudo-aleatoria a partir de una semilla, a la par que simula una función Hash, fue un claro ejemplo de cómo la teoría se traduce en práctica; Las pruebas realizadas, con sus entradas y salidas, permitieron valida la implementación y confirmar que el enfoque teórico era el adecuado para la solución del problema.

Por último, esta práctica resaltó el valor de los conceptos aprendidos como herramientas esenciales para el desarrollo de aplicaciones

Referencias

[1] GeeksforGeeks. *Collections in Java*. 2025. URL: https://www.geeksforgeeks.org/java/collections-in-java-2/.

[2] NIST. *Cryptographic Hash Functions*. 2025. URL: https://csrc.nist.gov/glossary/term/cryptographic_hash_function.

[3] Oracle. *Java Platform SE 8: Random Class Documentation*. 2014. URL: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html.