	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios 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): 320065570
425133840
423020252
322229916
425032224

No. de lista o brigada: 3

Semestre: 2026-1

Fecha de entrega: 27/08/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

En el ámbito del desarrollo de software, la falta de mecanismos para garantizar la integridad de los datos representa una desventaja crítica en el contexto actual. El problema central es que, sin una función que genere un identificador único (hash) para cada conjunto de datos, no existe forma confiable de verificar si la información ha sido alterada, ya sea por errores de transmisión, corrupción o manipulación malintencionada. Esto se convierte en un problema especialmente relevante en aplicaciones que manejan información sensible o requieren validación de autenticidad, ya que compromete la seguridad y la confiabilidad de todo el sistema. La motivación para resolver este problema radica en la ventaja de poder simular un mecanismo de integridad mediante funciones digestivas, lo que permite sentar las bases para aplicaciones más robustas en términos de seguridad. Implementar esta solución no solo ayuda a comprender el principio fundamental detrás de tecnologías ampliamente utilizadas, como las firmas digitales o la verificación de archivos, sino que también proporciona una herramienta para detectar cambios en los datos de manera eficiente y consistente, utilizando una salida pseudoaleatoria basada en una semilla. El objetivo de esta práctica es lograr la implementación de una función digestiva en Java que, a partir de una cadena ingresada desde el método main, genere un hash único simulando el comportamiento de los algoritmos criptográficos reales. Se espera reforzar el entendimiento de las clases de uso general para el manejo de cadenas, conversión de datos y generación de valores, así como demostrar la utilidad de estas herramientas en la creación de sistemas que requieran validación de integridad de información.

2. Marco Teórico

El marco teórico de esta solución se sustenta en tres pilares conceptuales de Java: las estructuras de datos dinámicas, los principios de las funciones hash y la generación de números pseudoaleatorios. La clase ArrayList pertenece al marco de las Colecciones de Java y proporciona una estructura de datos de lista ajustable, útil para almacenar y recorrer un número indeterminado de elementos, como los argumentos de línea de comandos [1]. La clase HashMap, también parte del framework de Colecciones, implementa una tabla hash que almacena pares clave-valor. Su uso es fundamental para asociar un hash único (clave) con su texto original correspondiente (valor), demostrando la capacidad de estas estructuras para realizar búsquedas eficientes. El concepto de función digestiva o hash es central en la informática para verificar la integridad de los datos. Una función hash ideal es determinista, de manera que una misma entrada siempre produce la misma salida; y está diseñada para que sea computacionalmente inviable revertir el proceso o encontrar dos entradas distintas que produzcan el mismo hash (colisión) [2]. Para simular este comportamiento, se utiliza un generador de números pseudoaleatorios (PRNG) de la clase Random. La característica clave es que, al inicializarse con la misma semilla, genera una secuencia de números idéntica y reproducible. Esto permite emular el determinismo de una función hash real: una cadena de entrada específica (convertida en una semilla) siempre producirá la misma secuencia numérica y, por ende, el mismo "hash"simulado.

3. Desarrollo

La implementación se realizó mediante un flujo que procesa los argumentos ingresados desde la consola. Inicialmente, el programa verifica la presencia de argumentos. De no existir, finaliza mostrando un mensaje indicativo del uso correcto. De lo contrario, cada argumento se almacena en un objeto ArrayList para facilitar su manejo iterativo. A continuación, se procesa cada elemento de la lista. Para cada cadena de texto, se invoca una función digestiva simulada. Internamente, esta función calcula una semilla numérica mediante la suma de los valores enteros (Unicode) de cada carácter de la cadena. Dicha semilla se utiliza para inicializar una instancia de Random. Seguidamente, se genera una secuencia de 32 valores enteros entre 0 y 15. Cada valor se convierte a su representación hexadecimal (un solo carácter) y se concatena para formar una cadena resultante de 32 caracteres, que se retorna como el hash. Cada par (hash, texto original) se inserta en un HashMap, usando el hash como clave. Entonces, el usuario, al introducir los argumentos como, por ejemplo, "hola mundo", obtendría una salida similar a la siguiente:

```
Resultados del HashMap:
=====================
Hash: 8f3a7b2c1d0e9f8a7b6c5d4e3f2a1b0c ->Texto original: hola
Hash: a5b9c8d7e6f5a4b3c2d1e0f9a8b7c6d ->Texto original: mundo
```

4. Resultados

A continuación se presentan los resultados obtenidos de la ejecución del programa, demostrando el correcto funcionamiento de la implementación de la función digestiva simulada y el manejo de las estructuras de datos.

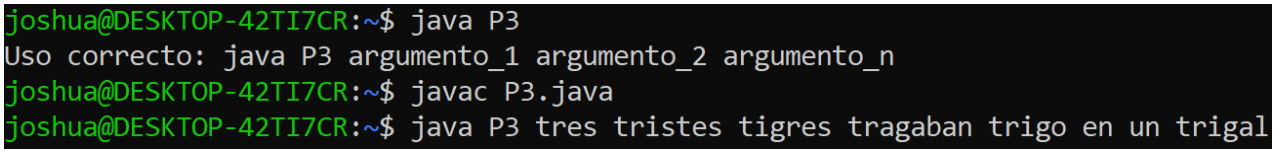

Figura 1: Compilación y ejecución del programa

Se muestra la compilación del archivo `P3.java` mediante el comando `javac`, seguida de la ejecución del programa sin argumentos. El sistema responde adecuadamente mostrando el mensaje de uso correcto: `Uso correcto: java P3 argumento_1 argumento_2 argumento_n`, lo que confirma que la validación de argumentos funciona según lo esperado.

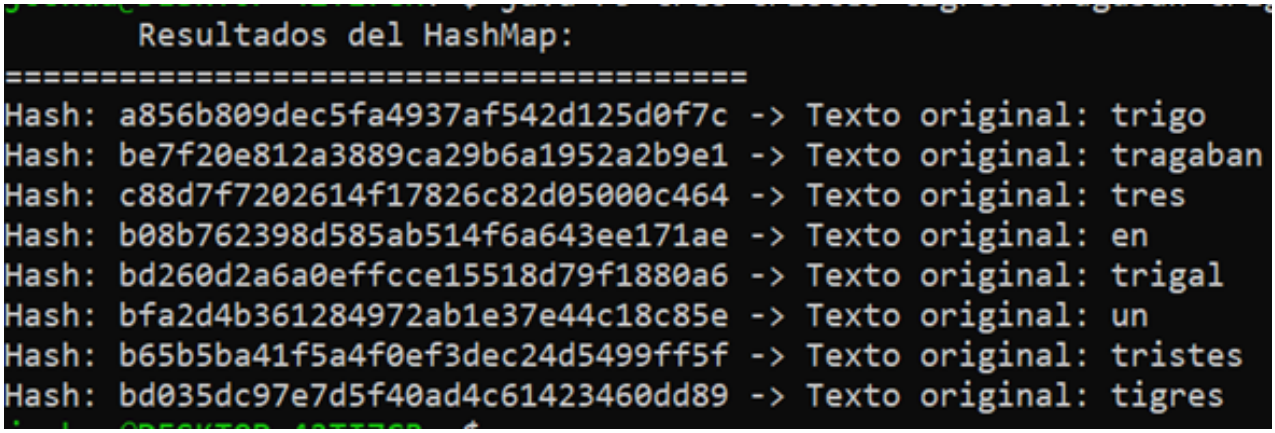

Figura 2: Resultados de la función hash simulada

Se presenta la salida generada al ejecutar el programa con los argumentos: `tres tristes tigres tragaban trigo en un trigal`. El programa procesa cada palabra individualmente mediante la función digestiva simulada, generando para cada una

un hash único de 32 caracteres hexadecimales. Los resultados se muestran correctamente organizados en forma de tabla, donde cada hash aparece asociado a su texto original correspondiente, demostrando el adecuado funcionamiento del `HashMap` para almacenar y presentar los pares clave-valor.

5. Conclusiones

Los principios de las estructuras de datos en Java pueden ser utilizados para simular mecanismos de seguridad esenciales, como lo son las funciones digestivas o hash; la practica que hemos tenido la oportundidad de elaborar en este caso, nos ha permitido comprender esto mismo. A través de la implementación de una función digestiva, se reforzó el uso de clases vistas en el marco Teorico como ArrayList, HashMap y Random, demostrando su importancia en la manipulación de cadenas, la asociación de datos y la generación de valores pseudoaleatorios. El algoritmo desarrollado cumplió el objetivo de ilustrar cómo es posible garantizar, la integridad de la información y detectar alteraciones en los datos. En este sentido, la práctica no solo contribuyó al fortalecimiento de conceptos previos que habiamos manejado, sino que también evidenció la relevancia de la validación de autenticidad en el desarrollo de aplicaciones seguras y confiables.

Referencias

[1] Oracle. *The Java Tutorials - Collections*. Oracle Help Center.

[2] Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons.