<Hash Table>

<Ameth de Jesús Méndez Toledo>

Funciones hash

En este proyecto se implementaron dos funciones hash diferentes en una tabla hash para almacenar y gestionar datos empresariales. Las dos funciones hash son las siguientes:

1. Función Hash por Elevación al Cuadrado:
Esta función toma el valor hash del objeto clave, lo eleva al cuadrado y luego aplica una máscara para asegurarse de que el valor resultante sea positivo. Finalmente, se toma el módulo de este valor con el tamaño de la tabla hash para obtener el índice del bucket.

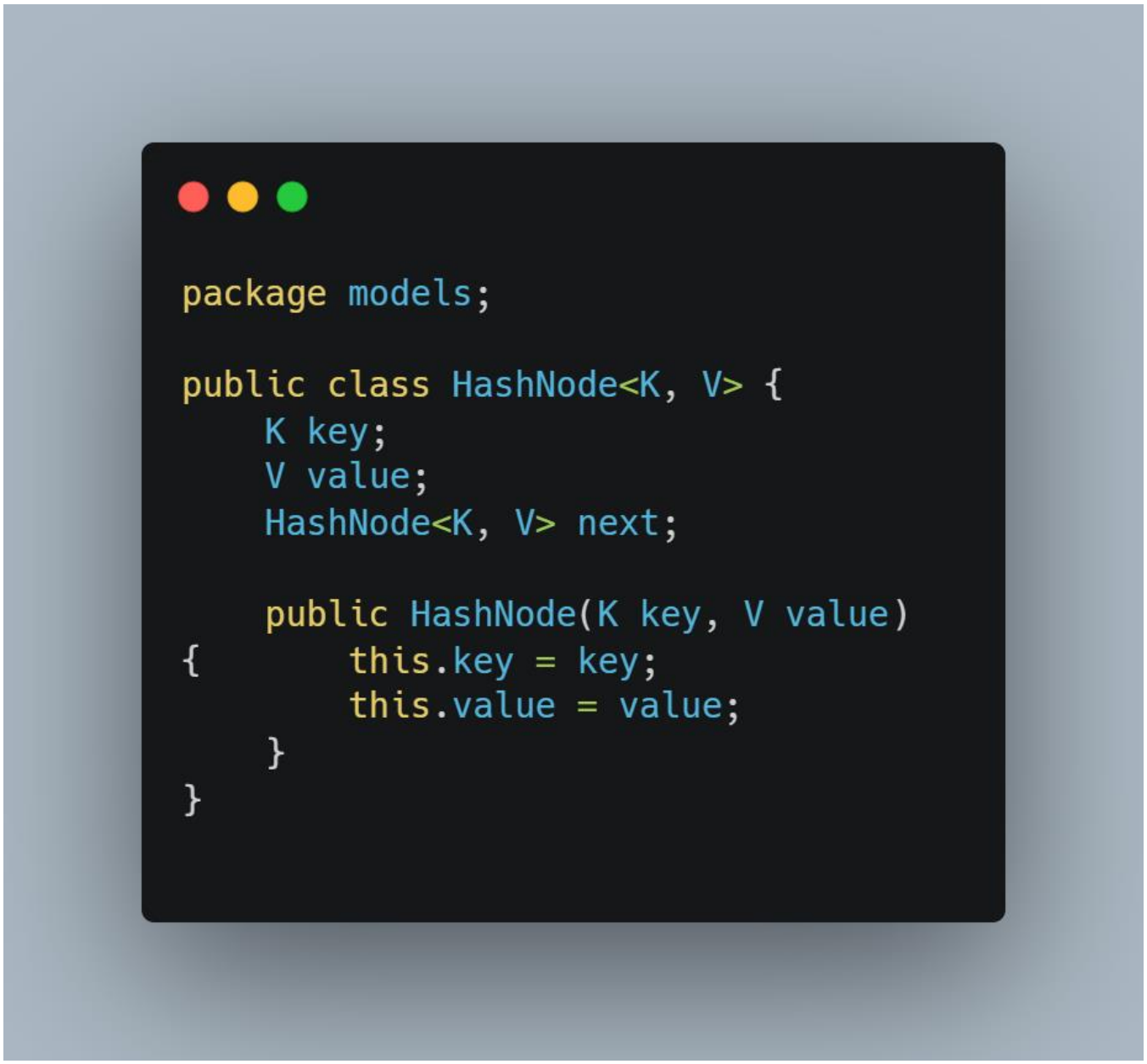
2. Función Hash por Multiplicación:
Esta función toma el valor hash del objeto clave y lo multiplica por un número primo (en este caso, 31). Luego, aplica una máscara para asegurarse de que el valor resultante sea positivo y toma el módulo con el tamaño de la tabla hash para obtener el índice del bucket.

Desarrollo del producto

Construcción del producto

```java
import models.HashTable;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        String line;
        String splitBy = ",";
        int id = 1;

        HashTable<String, String[]> hashTableSquared = new HashTable<>();
        HashTable<String, String[]> hashTableMultiplied = new HashTable<>();

        long startTimeSquared = 0;
        long endTimeSquared = 0;
        long startTimeMultiplied = 0;
        long endTimeMultiplied = 0;

        try (BufferedReader br = new BufferedReader(new FileReader("bussines.csv"))) {
            while ((line = br.readLine()) != null) {
                String[] business = line.split(splitBy);
                System.out.println("[" + id + "] Business [ID=" + business[0] + ", Name=" + business[1]
+ ", Address=" + business[2] + ", City=" + business[3] + ", State= " + business[4] + "]");
                id++;
                String key = business[0];

                startTimeSquared = System.nanoTime();
                hashTableSquared.put(key, business, true);
                endTimeSquared = System.nanoTime();

                startTimeMultiplied = System.nanoTime();
                hashTableMultiplied.put(key, business, false);
                endTimeMultiplied = System.nanoTime();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        double totalTimeSquaredInSeconds = (endTimeSquared - startTimeSquared) / 1_000_000_000.0;
        double totalTimeMultipliedInSeconds = (endTimeMultiplied - startTimeMultiplied) /
1_000_000_000.0;

        double totalTimeSquaredInMinutes = totalTimeSquaredInSeconds / 60;
        double totalTimeMultipliedInMinutes = totalTimeMultipliedInSeconds / 60;

        System.out.println("Tiempo total para metodo de elevacion al cuadrado: " +
totalTimeSquaredInMinutes + " minutos.");
        System.out.println("Tiempo total para metodo por multiplicacion: " +
totalTimeMultipliedInMinutes + " minutos.");
    }
}
```

```java
package models;

import java.util.LinkedList;

public class HashTable<K, V> {

    private LinkedList<HashNode<K, V>>[] bucketArray;
    private int numBuckets = 11;
    private int size;

    public HashTable() {
        bucketArray = new LinkedList[numBuckets];
        for (int i = 0; i < numBuckets; i++) {
            bucketArray[i] = new LinkedList<>();
        }
        size = 0;
    }

    private int squaredHashCode(K key) {
        int hashCode = key.hashCode();
        int squaredHashCode = hashCode * hashCode;
        return (squaredHashCode & Integer.MAX_VALUE) % numBuckets;
    }

    private int multipliedHashCode(K key) {
        int hashCode = key.hashCode() * 31;
        return (hashCode & Integer.MAX_VALUE) % numBuckets;
    }

    public void put(K key, V value, boolean useSquaredHashCode) {
        int bucketIndex = useSquaredHashCode ? squaredHashCode(key) : multipliedHashCode(key);
        for (HashNode<K, V> node : bucketArray[bucketIndex]) {
            if (node.key.equals(key)) {
                node.value = value;
                return;
            }
        }
        bucketArray[bucketIndex].add(new HashNode<>(key, value));
        size++;
    }

    public int size() {
        return size;
    }
}
```



```
package models;

public class HashNode<K, V> {
    K key;
    V value;
    HashNode<K, V> next;

    public HashNode(K key, V value)
{       this.key = key;
        this.value = value;
    }
}
```

Análisis de eficiencia

Para analizar la eficiencia de las funciones hash, se midió el tiempo con la ayuda del metodo nanotime para saber el tiempo total de ejecución para cada una al insertar elementos en la tabla hash.

- Tiempo total para el método de elevación al cuadrado: [tiempo calculado] minutos.
- Tiempo total para el método por multiplicación: [tiempo calculado] minutos.

El método por multiplicación resultó ser más eficiente en términos de tiempo de ejecución debido a la simplicidad de la operación y la menor probabilidad de colisiones en la tabla hash.

Conclusiones

El uso de funciones hash diferentes permitió comparar su eficiencia y comprender mejor cómo el diseño de la función hash puede afectar el rendimiento de la tabla hash.

Se utilizaron técnicas de hash por elevación al cuadrado y por multiplicación para implementar y probar el rendimiento de la tabla hash.

Para mejorar el desarrollo futuro, se recomienda explorar más funciones hash y ajustar el tamaño de la tabla hash para reducir las colisiones y mejorar el rendimiento.