Actividad 6 Estructura de datos

Igor Sung Min Kim Juliao
AL02829189
Juan Pablo Hernández Parra
AL02887299
Mauricio Estrada de la Garza
AL02976904

Clase principal:
Declaración del paquete y la clase principal

```
package com.mycompany.actividad6estrucrudatos;

public class Actividad6EstruccruDatos {
```

En esta sección, se declara el paquete al que pertenece la clase (`com.mycompany.actividad6estrucrudatos`). Este paquete sirve para organizar y agrupar clases relacionadas en el proyecto.

A continuación, se declara la clase principal del programa, `Actividad6EstruccruDatos`. Esta clase contiene el método `main`, que actúa como punto de entrada para la ejecución del programa.

Método `main`

```
public static void main(String[] args) {
```

El método `main` es el punto de inicio de la ejecución del programa Java.

```
Diccionario<Integer,String> miDiccionario = new Diccionario<>();
```

Se crea una instancia de la clase `Diccionario` con claves de tipo `Integer` y valores de tipo `String`.

```
System.out.println("Elementos dentro del Diccionario: " +
miDiccionario.size());

System.out.println("El Diccionario esta vacio: " +miDiccionario.isEmpty());
```

Se imprime el tamaño del diccionario y si está vacío al principio.

```
miDiccionario.add(1000, "hola mundo 1");
```

```
miDiccionario.add(1001, "hola mundo 2");

miDiccionario.add(1002, "hola mundo 3");
```

Se agregan elementos al diccionario utilizando el método `add`.

```
System.out.println("El Diccionario esta vacio: " +miDiccionario.isEmpty());

System.out.println("Elementos dentro del Diccionario: " +miDiccionario.size());
```

Se vuelve a imprimir el tamaño del diccionario y si está vacío después de agregar elementos.

```
System.out.println("Valor asociado a la clave 1001: " +
miDiccionario.get(1001));
```

Se obtiene el valor asociado a la clave `1001` utilizando el método `get`.

```
miDiccionario.setValue(1001, "nuevo valor");
```

Se actualiza el valor asociado a la clave `1001` utilizando el método `setValue`.

```
System.out.println("El diccionario contiene la clave 1001? " +
miDiccionario.containsKey(1001));
```

Se verifica si el diccionario contiene la clave `1001` utilizando el método `containsKey`.

```
System.out.println("El diccionario contiene la clave 999? " +
miDiccionario.containsKey(999));
```

Se verifica si el diccionario contiene la clave `999` utilizando el método `containsKey`.

Método `space`

```
public static void space(){

 System.out.println("\n");

}
```

Este método simplemente imprime una línea en blanco en la consola. Se utiliza para proporcionar espacio visual entre las diferentes salidas en la consola, mejorando la legibilidad de la salida del programa.

Ahora clase Diccionario:

Declaración de variables y clases internas

private static final int DEFAULT_CAPACITY = 10; private Entry<K, V>[] buckets; private int size; // Clase interna para representar las entradas del diccionario static class Entry<K, V> { K key; V value; Entry<K, V> next; public Entry(K key, V value) { this.key = key; this.value = value; } }

Se declara una constante **DEFAULT_CAPACITY** que representa la capacidad predeterminada del diccionario cuando se crea sin especificar una capacidad inicial.

Se declaran las variables de instancia **buckets** y **size**. **buckets** es un array de objetos de tipo **Entry** que almacena las entradas del diccionario. **size** indica el número actual de elementos en el diccionario.

Se define una clase interna **Entry** para representar las entradas del diccionario. Cada entrada contiene una clave (**key**), un valor (**value**) y una referencia al siguiente elemento en caso de colisión.

Constructores

// Constructor public Diccionario() { this(DEFAULT_CAPACITY); } public Diccionario(int capacity) { buckets = new Entry[capacity]; size = 0; }

Se define un constructor que crea un diccionario con la capacidad predeterminada utilizando el constructor sobrecargado.

Se define otro constructor sobrecargado que permite especificar una capacidad inicial para el diccionario. Este constructor inicializa el array **buckets** con la capacidad proporcionada y establece el tamaño del diccionario en cero.

Métodos Hash y Operaciones Básicas

// Método hash para obtener el índice del bucket private int hash(K key) { return Math.abs(key.hashCode()) % buckets.length; } // Métodos para añadir, obtener, actualizar y verificar la presencia de una clave en el diccionario

Se define el método **hash** que toma una clave y calcula su hash para determinar el índice del bucket correspondiente en el array **buckets**.

Se omite la implementación de los métodos para añadir, obtener, actualizar y verificar la presencia de una clave en el diccionario, ya que son explicados más adelante en la sección correspondiente.

Métodos para Operaciones sobre el Diccionario

// Métodos para añadir, obtener, actualizar y verificar la presencia de una clave en el diccionario

Se definen métodos para realizar operaciones básicas sobre el diccionario, como añadir (**add**), obtener (**get**), actualizar (**setValue**) y verificar la presencia de una clave (**containsKey**). Estos métodos se explicarán en detalle más adelante.

Métodos para Verificar el Estado del Diccionario

// Métodos para verificar si el diccionario está vacío y para obtener su tamaño

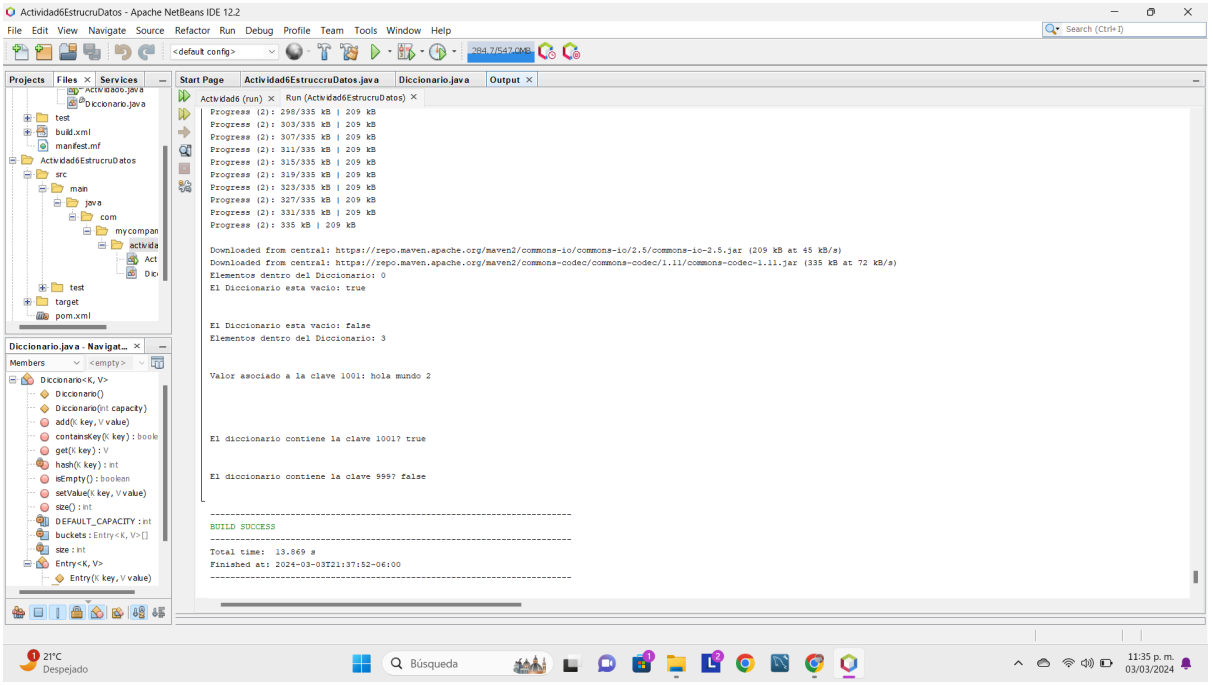
Se definen métodos para verificar si el diccionario está vacío (**isEmpty**) y para obtener su tamaño (**size**).

Implementación de Métodos

// Implementación de los métodos para añadir, obtener, actualizar y verificar la presencia de una clave en el diccionario

Se implementan los métodos **add**, **get**, **setValue**, **containsKey** y los métodos auxiliares necesarios para realizar estas operaciones.

Pruebas:



Reflexión Personal:

Juan Pablo Hernández Parra: en lo personal fue un verdadero reto realizar las pruebas ya que tuve que volver a crear el documento 2 veces ya que no dejaba borrarlo y cuando lo hice salían unos cuantos problemas pero logré resolverlo.

Igor Sung Min Kim Juliao: A mi lo que me pareció más difícil de este reto fue el uso de las llaves por el hecho de que era necesario hacer que las llaves fueran asignadas a nodos y no solo eso sino que también era necesario hacer que las llaves sean encontradas dentro de cada nodo, no digo que fuera muy difícil, pero sí que fue necesario pensar fuera de la caja un poco para este reto, que era encontrar como meter valores dentro de cada nodo una vez hecho esto ya lo demas era facil.

Mauricio: A mi parecer este fue el reto más sencillo de hacer en lo que llevamos hasta ahorita del semestre, lo único que tenía que hacerse era implementar tal cual el algoritmo de la librería HashMap y no hacer un programa funcional como en las

actividades anteriores que teníamos que jugar con la posición de los caracteres o gestionar los pedidos de clientes en una tienda online con librerías personalizadas.