INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	**ESTRUCTURA DE DATOS Y ALGORITMOS**				
TÍTULO DE LA PRÁCTICA:	**HASHING**				
NÚMERO DE PRÁCTICA:	**09**	**AÑO LECTIVO:**	**2025 – A**	**NRO. SEMESTRE:**	**Tercero III**
FECHA DE PRESENTACIÓN	**15/07/2025**	**HORA DE PRESENTACIÓN**	**23:59**		
INTEGRANTE (s): Davila Flores Mathias Dario				**NOTA:**	
DOCENTE(s): ● **Mg. Ing. Rene Alonso Nieto Valencia.** ● **ENLACE GITHUB:** https://github.com/mathiasddf/LabsEDA					

SOLUCIÓN Y RESULTADOS

I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS

a. Ejercicios Resueltos:

- **IMPLEMENTACIÓN HASH CERRADO.**
  - i. **Ejercicio 1:** Implementación clase Registro Genérico, clave entera y valor de tipo <E>

```java
package Lab9.Ej_resueltos.Hash_Cerrado;

public class Register<E> implements Comparable<Register<E>> {
    private int key;
    private E value;
    public Register(int key, E value) {
        this.key = key;
        this.value = value;
    }
    public int getKey() {
        return key;
    }
    public E getValue() {
        return value;
    }
    public void setValue(E value) {
        this.value = value;
    }
    @Override
    public int compareTo(Register<E> other) {
        return Integer.compare(this.key, other.key);
    }
    @Override
    public String toString() {
        return key + ": " + value;
    }
}
```

ii. **Ejercicio 2:** Implementación clase HashClosed genérico, para representar el sondeo lineal con la clase Element y las operaciones para manipular los elementos de la tabla (agregar, buscar, eliminar y mostrar tabla).

```java
    private int hash(E key) {
        return Math.abs(key.hashCode()) % table.length;
    }

    public boolean insert(E key) {
        if (size == table.length) {
        System.out.println(" Tabla llena. No se puede insertar: " + key);
        return false;
        }

        int index = hash(key);
        int start = index;

        do {
        Element<E> element = table[index];
        if (element == null || element.isDeleted()) {
            table[index] = new Element<>(key);
            size++;
            System.out.println(" Insertado: " + key + " en índice " + index);
            return true;
        }
        if (!element.isDeleted() && element.getValue().equals(key)) {
            System.out.println(" Clave duplicada: " + key);
            return false;
        }
        index = (index + 1) % table.length;
        } while (index != start);

        System.out.println(" No se pudo insertar: " + key);
        return false;
    }
```

```java
    public boolean search(E key) {
        int index = hash(key);
        int start = index;

        do {
        Element<E> element = table[index];
        if (element == null) return false;
        if (!element.isDeleted() && element.getValue().equals(key)) return true;
        index = (index + 1) % table.length;
        } while (index != start);

        return false;
    }

    public boolean delete(E key) {
        int index = hash(key);
        int start = index;

        do {
        Element<E> element = table[index];
        if (element == null) return false;
        if (!element.isDeleted() && element.getValue().equals(key)) {
            element.markDeleted();
            size--;
            System.out.println(" Eliminado: " + key + " en índice " + index);
            return true;
        }
        index = (index + 1) % table.length;
        } while (index != start);

        System.out.println(" No encontrado para eliminar: " + key);
        return false;
```

```java
package Lab9.Ej_resueltos.Hash_Cerrado;

public class Element<T> {
    private T value;
    private boolean deleted;

    public Element(T value) {
        this.value = value;
        this.deleted = false;
    }

    public T getValue() {
        return value;
    }

    public boolean isDeleted() {
        return deleted;
    }

    public void markDeleted() {
        this.deleted = true;
    }

    @Override
    public String toString() {
        return deleted ? "X" : value.toString();
    }
}
```

iii. **Ejercicio 3 :** Implementar la Clase TestHashClosed genérica, para probar la implementación con la agregación de los elementos asociados con diferentes tipos de datos

- Agregar los elementos: 100, 5, 14, 15, 22, 16, 17, 32, 13, 32, 100.
- Mostrar tabla hash resultante.
- Buscar los elementos: 32, 200.
- Eliminar los elementos. 17, 100.
- Mostrar tabla hash resultante.

```
cal\Temp\cp_1jk95xt18j6qk7fci1fkehiym.argfile' 'Lab9.Ej_resueltos.Hash_Cerrado.TestHashClosed'
=== Inserción de elementos ===
 Insertado: 100 en índice 1
 Insertado: 5 en índice 5
 Insertado: 14 en índice 3
 Insertado: 15 en índice 4
 Insertado: 22 en índice 0
 Insertado: 16 en índice 6
 Insertado: 17 en índice 7
 Insertado: 32 en índice 10
 Insertado: 13 en índice 2
 Clave duplicada: 32
 Clave duplicada: 100

=== Tabla luego de inserciones ===
Tabla Hash:
0: 22
1: 100
2: 13
3: 14
4: 15
5: 5
6: 16
7: 17
8: ?
9: ?
10: 32

=== Búsqueda de elementos ===
Buscar 32:  Encontrado
Buscar 200:  No encontrado
```

```
=== Eliminación de elementos ===
 Eliminado: 17 en índice 7
 Eliminado: 100 en índice 1

=== Tabla luego de eliminaciones ===
Tabla Hash:
0: 22
1: X
2: 13
3: 14
4: 15
5: 5
6: 16
7: X
8: ?
9: ?
10: 32

=== Inserción tras eliminaciones (prueba adicional) ===
 Insertado: 77 en índice 1
Tabla Hash:
0: 22
1: 77
2: 13
3: 14
4: 15
5: 5
6: 16
7: X
8: ?
9: ?
10: 32
```

- **IMPLEMENTACIÓN HASH ABIERTO**
  - **i. Ejercicio 1:** Implementación clase Registro Genérico, clave entera y valor de tipo <E>

```java
public class Register<E> implements Comparable<Register<E>> {
    private int key;
    private E value;
    private boolean deleted;
    public Register(int key, E value) {
        this.key = key;
        this.value = value;
        this.deleted = false;
    }
    public int getKey() {
        return key;
    }
    public E getValue() {
        return value;
    }
    public void setValue(E value) {
        this.value = value;
    }
    public boolean isDeleted() {
        return deleted;
    }
    public void delete() {
        this.deleted = true;
    }
    @Override
    public int compareTo(Register<E> other) {
        return Integer.compare(this.key, other.key);
    }
    @Override
    public String toString() {
        return (deleted ? "[ELIMINADO] " : "") + key + ":" + value;
    }
}
```

  - **ii. Ejercicio 2:** Implementación clase HashOpened genérico, para representar encadenamiento o hash abierto con la clase Element, utilizando listas enlazadas y las operaciones para manipular los elementos de la tabla (agregar, buscar, eliminar y mostrar tabla).

```
    public void insert(Element<E> elem) {
        int index = hash(elem.getKey());

        for (Element<E> e : table[index]) {
        if (e.getKey() == elem.getKey() && !e.isDeleted()) {
            System.out.println(" Clave duplicada: " + elem.getKey());
            return;
        }
        }

        table[index].add(elem);
        System.out.println(" Insertado: " + elem);
    }

    public void delete(int key) {
        int index = hash(key);

        for (Element<E> e : table[index]) {
        if (e.getKey() == key && !e.isDeleted()) {
            e.delete();
            System.out.println(" Eliminado lógicamente: " + key);
            return;
        }
        }

        System.out.println(" Clave no encontrada: " + key);
    }

```

```
    public Element<E> search(int key) {
        int index = hash(key);

        for (Element<E> e : table[index]) {
        if (e.getKey() == key && !e.isDeleted()) {
            return e;
        }
        }

        return null;
    }

    public void showTable() {
        System.out.println(x:"\n--- Estado de la Tabla Hash (Abierto) ---");
        for (int i = 0; i < table.length; i++) {
        System.out.print(i + ": ");
        if (table[i].isEmpty()) {
            System.out.println(x:"[VACÍO]");
        } else {
            for (Element<E> e : table[i]) {
            System.out.print(e + " -> ");
            }
            System.out.println(x:"null");
        }
        }
    }
```

Lab9 > Ej_resueltos > Hash_Abierto > Element.java > ...

```java
public class Element<T> implements Comparable<Element<T>> {

    public Element(int key, T value) {
        this.key = key;
        this.value = value;
        this.deleted = false;
    }

    public int getKey() {
        return key;
    }

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }

    public boolean isDeleted() {
        return deleted;
    }

    public void delete() {
        this.deleted = true;
    }

    @Override
    public int compareTo(Element<T> other) {
        return Integer.compare(this.key, other.key);
    }

    @Override
    public String toString() {
        return (deleted ? "[ELIMINADO] " : "") + key + ":" + value;
    }
```

iii. Ejercicio 3: Implementar la Clase TestHashOpened genérica, para probar la implementación con la agregación de los elementos asociados con diferentes tipos de datos.

- El tamaño de la tabla es: 8.
- Agregar el elemento: clave: 5, valor: Pepe.
- Agregar el elemento: clave: 21, valor: Jesús.
- Agregar el elemento: clave: 19, valor: Juan.
- Agregar el elemento: clave: 16, valor: María.
- Agregar el elemento: clave: 21, valor: DUPLICADO.
- Mostrar tabla hash resultante.
- Buscar los elementos por clave: 5, 21.
- Eliminar los elementos por clave. 21, 100.
- Mostrar tabla hash resultante.

```
=== Inserciones ===
 Insertado: 5:Pepe
 Insertado: 21:Jesús
 Insertado: 19:Juan
 Insertado: 16:María
 Clave duplicada: 21

=== Mostrar tabla ===

--- Estado de la Tabla Hash (Abierto) ---
0: 16:María -> null
1: [VACÍO]
2: [VACÍO]
3: 19:Juan -> null
4: [VACÍO]
5: 5:Pepe -> 21:Jesús -> null
6: [VACÍO]
7: [VACÍO]

=== Búsquedas ===
Clave 5 encontrada: 5:Pepe
Clave 21 encontrada: 21:Jesús

=== Eliminaciones ===
 Eliminado lógicamente: 21
 Clave no encontrada: 100
```

```
=== Tabla después de eliminaciones ===

--- Estado de la Tabla Hash (Abierto) ---
0: 16:María -> null
1: [VACÍO]
2: [VACÍO]
3: 19:Juan -> null
4: [VACÍO]
5: 5:Pepe -> [ELIMINADO] 21:Jesús -> null
6: [VACÍO]
7: [VACÍO]

=== Inserción tras eliminación ===
 Insertado: 21:Nuevo Jesús

--- Estado de la Tabla Hash (Abierto) ---
0: 16:María -> null
1: [VACÍO]
2: [VACÍO]
3: 19:Juan -> null
4: [VACÍO]
5: 5:Pepe -> [ELIMINADO] 21:Jesús -> 21:Nuevo Jesús -> null
6: [VACÍO]
7: [VACÍO]
```

b. Ejercicios Propuestos:

i. **EJERCICIO 1:** Implementar mediante generalidad las clases y métodos genéricos de las actividades resueltas para el HASH CERRADO, incluyendo todos los métodos y clases para manejar las colisiones y representar el recorrido a la siguiente posición cuando exista colisión.

- **Breve explicación:**

  En el **Ejercicio 1**, se implementó una tabla hash utilizando la técnica de **hash cerrado** con sondeo lineal, bajo una estructura totalmente **genérica** mediante el uso de la clase HashClosed<E>. Esta estructura emplea arreglos de registros (Register<E>) donde las colisiones se manejan recorriendo linealmente la tabla hasta encontrar una posición vacía o reutilizable. Se desarrollaron los métodos fundamentales: insertar (insert), buscar (search), eliminar lógicamente (delete) y mostrar la tabla (showTable). Además, se asegura que no se ingresen claves duplicadas, y al eliminar, el espacio se marca como disponible para futuras inserciones. La implementación permite almacenar cualquier tipo de dato gracias a su diseño genérico.

- **Register.java**

```java
package Lab9.Ej_propuestos.Ejercicio1;

public class Register<E> implements Comparable<Register<E>> {
    private int key;
    private E value;
    private boolean deleted;

    public Register(int key, E value) {
        this.key = key;
        this.value = value;
        this.deleted = false;
    }

    public int getKey() { return key; }
    public E getValue() { return value; }
    public void setValue(E value) { this.value = value; }
    public boolean isDeleted() { return deleted; }
    public void delete() { this.deleted = true; }

    @Override
    public int compareTo(Register<E> o) {
        return Integer.compare(this.key, o.key);
    }

    @Override
    public String toString() {
        return deleted ? "[ELIMINADO] " + key + ":" + value : key + ":" + value;
    }
}
```

- **HashClosed.java**

```java
    public boolean insert(Register<E> reg) {
        if (size == table.length) {
        System.out.println(" Tabla llena. No se puede insertar: " + reg);
        return false;
        }

        int index = hash(reg.getKey());
        int start = index;

        do {
        Register<E> current = table[index];

        if (current == null || current.isDeleted()) {
            table[index] = reg;
            size++;
            System.out.println(" Insertado: " + reg + " en índice " + index);
            return true;
        }

        if (!current.isDeleted() && current.getKey() == reg.getKey()) {
            System.out.println(" Clave duplicada: " + reg.getKey());
            return false;
        }

        index = (index + 1) % table.length;

        } while (index != start);

        System.out.println(" No se pudo insertar: " + reg);
        return false;
    }
```

```java
    public Register<E> search(int key) {
        int index = hash(key);
        int start = index;

        do {
        Register<E> current = table[index];

        if (current == null) return null;
        if (!current.isDeleted() && current.getKey() == key) return current;

        index = (index + 1) % table.length;

        } while (index != start);

        return null;
    }
```

```java
    public boolean delete(int key) {
        int index = hash(key);
        int start = index;

        do {
        Register<E> current = table[index];

        if (current == null) return false;
        if (!current.isDeleted() && current.getKey() == key) {
            current.delete();
            size--;
            System.out.println(" Eliminado lógicamente: " + key);
            return true;
        }

        index = (index + 1) % table.length;

        } while (index != start);

        return false;
    }

    public void showTable() {
        System.out.println(x:"\n--- Tabla Hash Cerrado ---");
        for (int i = 0; i < table.length; i++) {
        System.out.print(i + ": ");
        if (table[i] == null) {
            System.out.println(x:"[VACÍO]");
        } else {
            System.out.println(table[i]);
        }
```

- **Resultados**

```
=== Inserciones ===
  Insertado: 5:Pepe en índice 5
  Insertado: 12:Ana en índice 6
  Insertado: 9:Luis en índice 2
  Insertado: 16:Pedro en índice 3
  Clave duplicada: 12

 --- Tabla Hash Cerrado ---
 0: [VACÍO]
 1: [VACÍO]
 2: 9:Luis
 3: 16:Pedro
 4: [VACÍO]
 5: 5:Pepe
 6: 12:Ana

 === Búsqueda ===
  Clave 12 encontrada ? Valor: Ana
  Clave 100 no encontrada

 === Eliminaciones ===
  Eliminado lógicamente: 12

 --- Tabla Hash Cerrado ---
 0: [VACÍO]
 1: [VACÍO]
 2: 9:Luis
 3: 16:Pedro
 4: [VACÍO]
 5: 5:Pepe
 6: [ELIMINADO] 12:Ana

 === Búsqueda después de eliminar ===
  Clave 12 no encontrada
```

ii. **EJERCICIO 2:** Implementar mediante generalidad las clases y métodos genéricos de las actividades resueltas para el HASH ABIERTO, incluyendo todos los métodos y clases para manejar las colisiones y representar el recorrido en las listas enlazadas cuando exista colisión.

- **Breve explicación:**

  En el **Ejercicio 2**, se diseñó una tabla hash con **hash abierto**, utilizando listas enlazadas para manejar colisiones. La clase HashOpened<E> trabaja con un arreglo de listas enlazadas (LinkedList<Register<E>>), donde cada posición del arreglo puede almacenar múltiples elementos que comparten el mismo valor de hash. Cuando se detecta una colisión, el nuevo elemento se agrega al final de la lista correspondiente. Esta solución también incluye los métodos para insertar, buscar, eliminar lógicamente y mostrar el contenido de la tabla, incluyendo todas las listas enlazadas. Al igual que en el caso anterior, se mantiene la generalidad en toda la implementación, permitiendo trabajar con distintos tipos de datos.

- **Register.java**

ab9 > Ej_propuestos > Ejercicio2 > Register.java > ...

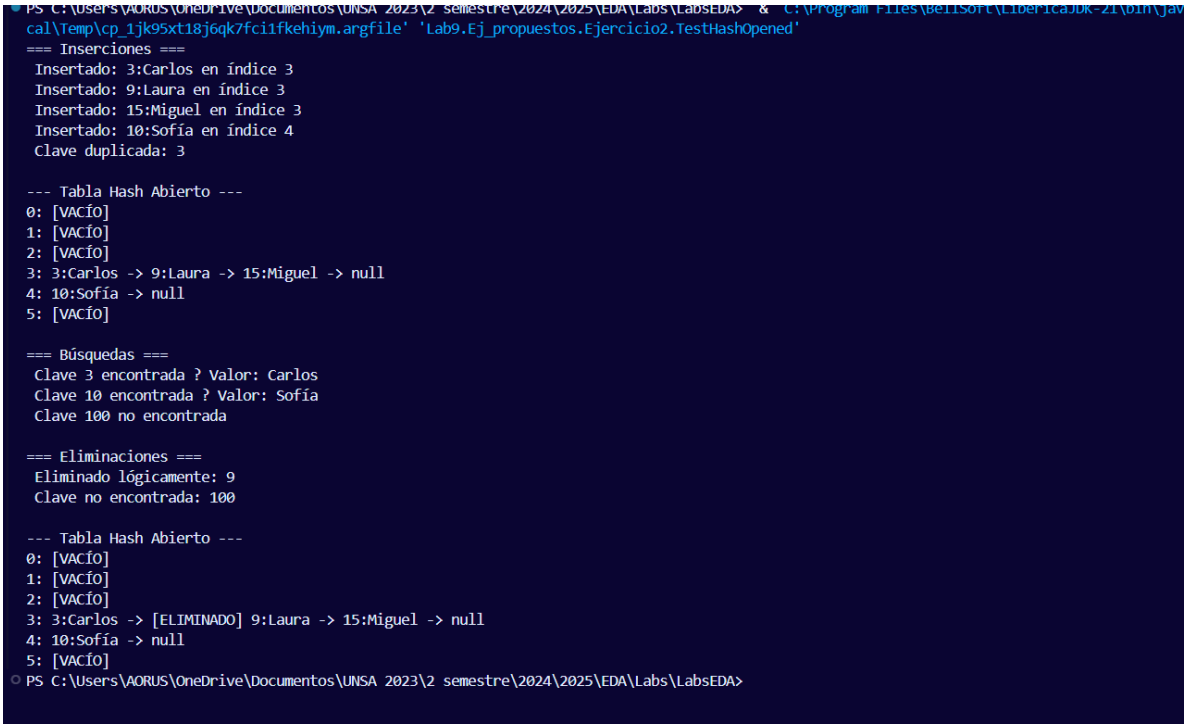
```java
package Lab9.Ej_propuestos.Ejercicio2;

public class Register<E> implements Comparable<Register<E>> {
    private int key;
    private E value;
    private boolean deleted;

    public Register(int key, E value) {
        this.key = key;
        this.value = value;
        this.deleted = false;
    }

    public int getKey() { return key; }
    public E getValue() { return value; }
    public void setValue(E value) { this.value = value; }
    public boolean isDeleted() { return deleted; }
    public void delete() { this.deleted = true; }

    @Override
    public int compareTo(Register<E> other) {
        return Integer.compare(this.key, other.key);
    }

    @Override
    public String toString() {
        return (deleted ? "[ELIMINADO] " : "") + key + ":" + value;
    }
}

```

- **HashOpened.java**

```java
    public boolean insert(Register<E> reg) {
        int index = hash(reg.getKey());

        for (Register<E> r : table[index]) {
        if (r.getKey() == reg.getKey() && !r.isDeleted()) {
            System.out.println(" Clave duplicada: " + reg.getKey());
            return false;
        }
        }

        table[index].add(reg);
        System.out.println(" Insertado: " + reg + " en índice " + index);
        return true;
    }

    public boolean delete(int key) {
        int index = hash(key);

        for (Register<E> r : table[index]) {
        if (r.getKey() == key && !r.isDeleted()) {
            r.delete();
            System.out.println(" Eliminado lógicamente: " + key);
            return true;
        }
        }

        System.out.println(" Clave no encontrada: " + key);
        return false;
    }
```

```java
    public Register<E> search(int key) {
        int index = hash(key);

        for (Register<E> r : table[index]) {
        if (r.getKey() == key && !r.isDeleted()) {
            return r;
        }
        }

        return null;
    }

    // Método genérico solicitado
    public E buscarValor(int key) {
        Register<E> result = search(key);
        return result != null ? result.getValue() : null;
    }

    public void showTable() {
        System.out.println(x:"\n--- Tabla Hash Abierto ---");
        for (int i = 0; i < table.length; i++) {
        System.out.print(i + ": ");
        if (table[i].isEmpty()) {
            System.out.println(x:"[VACÍO]");
        } else {
            for (Register<E> r : table[i]) {
            System.out.print(r + " -> ");
            }
            System.out.println(x:"null");
        }
        }
```

- **Resultados**



iii. **EJERCICIO 3:** Implementar Hash cerrado, con clases y métodos genéricos:

- Implementar todas las operaciones necesarias.
- Implementar una clase Test para probar los métodos y mostrar los resultados. utilizando clases y métodos genéricos, utilizando un menú de opciones para todas las operaciones del HASH CERRADO.

- **Breve explicación:**

El Ejercicio 3 extiende la implementación del hash cerrado incluyendo un menú interactivo por consola, a través de la clase TestHashClosedConMenu. Se conservan las operaciones fundamentales ya desarrolladas, y se agregan dos funciones útiles para el manejo dinámico de la tabla: isEmpty, que indica si hay elementos activos en la tabla, y clear, que permite vaciar la estructura completamente. Este menú permite al usuario realizar inserciones, búsquedas, eliminaciones, mostrar el estado actual de la tabla, verificar si está vacía o limpiarla. Todo se realiza usando clases y métodos genéricos, permitiendo aplicar la estructura con diferentes tipos de datos sin modificar la lógica del código.

- **Register.java**

```java
package Lab9.Ej_propuestos.Ejercicio3;

public class Register<E> implements Comparable<Register<E>> {
    private int key;
    private E value;
    private boolean deleted;

    public Register(int key, E value) {
        this.key = key;
        this.value = value;
        this.deleted = false;
    }

    public int getKey() { return key; }
    public E getValue() { return value; }
    public void setValue(E value) { this.value = value; }
    public boolean isDeleted() { return deleted; }
    public void delete() { this.deleted = true; }

    @Override
    public int compareTo(Register<E> o) {
        return Integer.compare(this.key, o.key);
    }

    @Override
    public String toString() {
        return deleted ? "[ELIMINADO] " + key + ":" + value : key + ":" + value;
    }
}
```

- **HashClosed.java**

```java
    public boolean insert(Register<E> reg) {
        if (size == table.length) {
        System.out.println(x:" Tabla llena. No se puede insertar.");
        return false;
        }

        int index = hash(reg.getKey());
        int start = index;

        do {
        Register<E> current = table[index];

        if (current == null || current.isDeleted()) {
            table[index] = reg;
            size++;
            System.out.println(" Insertado: " + reg + " en índice " + index);
            return true;
        }

        if (!current.isDeleted() && current.getKey() == reg.getKey()) {
            System.out.println(" Clave duplicada: " + reg.getKey());
            return false;
        }

        index = (index + 1) % table.length;

        } while (index != start);

        return false;
    }
```

```java
    public Register<E> search(int key) {
        int index = hash(key);
        int start = index;

        do {
        Register<E> current = table[index];
        if (current == null) return null;
        if (!current.isDeleted() && current.getKey() == key) return current;
        index = (index + 1) % table.length;
        } while (index != start);

        return null;
    }

    public boolean delete(int key) {
        int index = hash(key);
        int start = index;

        do {
        Register<E> current = table[index];
        if (current == null) return false;
        if (!current.isDeleted() && current.getKey() == key) {
            current.delete();
            size--;
            System.out.println(" Eliminado lógicamente: " + key);
            return true;
        }
        index = (index + 1) % table.length;
        } while (index != start);

        return false;
    }
```

```java
    public boolean isEmpty() {
        return size == 0;
    }

    public void clear() {
        for (int i = 0; i < table.length; i++) {
        table[i] = null;
        }
        size = 0;
        System.out.println(x:" Tabla vaciada.");
    }

    public void showTable() {
        System.out.println(x:"\n--- Estado actual de la Tabla Hash (Cerrado) ---");
        for (int i = 0; i < table.length; i++) {
        System.out.print(i + ": ");
        if (table[i] == null) {
            System.out.println(x:"[VACÍO]");
        } else {
            System.out.println(table[i]);
        }
```

- **Resultados**

```
=== MENÚ - HASH CERRADO ===
1. Insertar
2. Buscar
3. Eliminar
4. Mostrar tabla
5. ¿Hash vacio?
6. Limpiar tabla
0. Salir
Elige una opción: 1
Clave (int): 5
Valor (String): alex
 Insertado: 5:alex en índice 5

=== MENÚ - HASH CERRADO ===
1. Insertar
2. Buscar
3. Eliminar
4. Mostrar tabla
5. ¿Hash vacio?
6. Limpiar tabla
0. Salir
Elige una opción: 4

--- Estado actual de la Tabla Hash (Cerrado) ---
0: [VACÍO]
1: [VACÍO]
2: [VACÍO]
3: [VACÍO]
4: [VACÍO]
5: 5:alex
6: [VACÍO]
7: [VACÍO]
8: [VACÍO]
9: [VACÍO]
```

```
PROBLEMAS 16  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS        Run: TestHashClosed

0. Salir
Elige una opción: 1
Clave (int): 17
Valor (String): cinco
 Insertado: 17:cinco en índice 9

=== MENÚ - HASH CERRADO ===
1. Insertar
2. Buscar
3. Eliminar
4. Mostrar tabla
5. ¿Hash vacio?
6. Limpiar tabla
0. Salir
Elige una opción: 4

--- Estado actual de la Tabla Hash (Cerrado) ---
0: [VACÍO]
1: [VACÍO]
2: [VACÍO]
3: [VACÍO]
4: [VACÍO]
5: 5:uno
6: 15:dos
7: 25:tres
8: 7:cuatro
9: 17:cinco

=== MENÚ - HASH CERRADO ===
1. Insertar
2. Buscar
3. Eliminar
4. Mostrar tabla
5. ¿Hash vacio?
6. Limpiar tabla
0. Salir
Elige una opción:
```

iv. **EJERCICIO 4:** Implementar Hash abierto, con clases y métodos genéricos:

- Implementar todas las operaciones necesarias.
- Implementar una clase Test para probar los métodos y mostrar los resultados. utilizando clases y métodos genéricos, utilizando un menú de opciones para todas las operaciones del HASH ABIERTO.

- **Breve explicación:**

  En el **Ejercicio 4**, se aplicó el mismo enfoque del ejercicio anterior pero para la estructura de **hash abierto**. La clase TestHashOpenedConMenu ofrece una interfaz de usuario en consola para probar las operaciones de la tabla HashOpened<E>, incluyendo inserción, búsqueda, eliminación, visualización, verificación de vaciado e inicialización de la tabla. Las colisiones se manejan eficazmente mediante listas enlazadas, y el diseño generalizado permite una reutilización flexible del código con distintos tipos de datos. Este ejercicio refuerza la importancia del uso de métodos y clases genéricas en estructuras de datos adaptables y reutilizables.

- **Register.java**

Lab9 > Ej_propuestos > Ejercicio4 > Register.java > {} Lab9.Ej_propuestos.Ejercicio4

```java
package Lab9.Ej_propuestos.Ejercicio4;

public class Register<E> implements Comparable<Register<E>> {
    private int key;
    private E value;
    private boolean deleted;

    public Register(int key, E value) {
        this.key = key;
        this.value = value;
        this.deleted = false;
    }

    public int getKey() { return key; }
    public E getValue() { return value; }
    public void setValue(E value) { this.value = value; }
    public boolean isDeleted() { return deleted; }
    public void delete() { this.deleted = true; }

    @Override
    public int compareTo(Register<E> other) {
        return Integer.compare(this.key, other.key);
    }

    @Override
    public String toString() {
        return (deleted ? "[ELIMINADO] " : "") + key + ":" + value;
    }
}

```

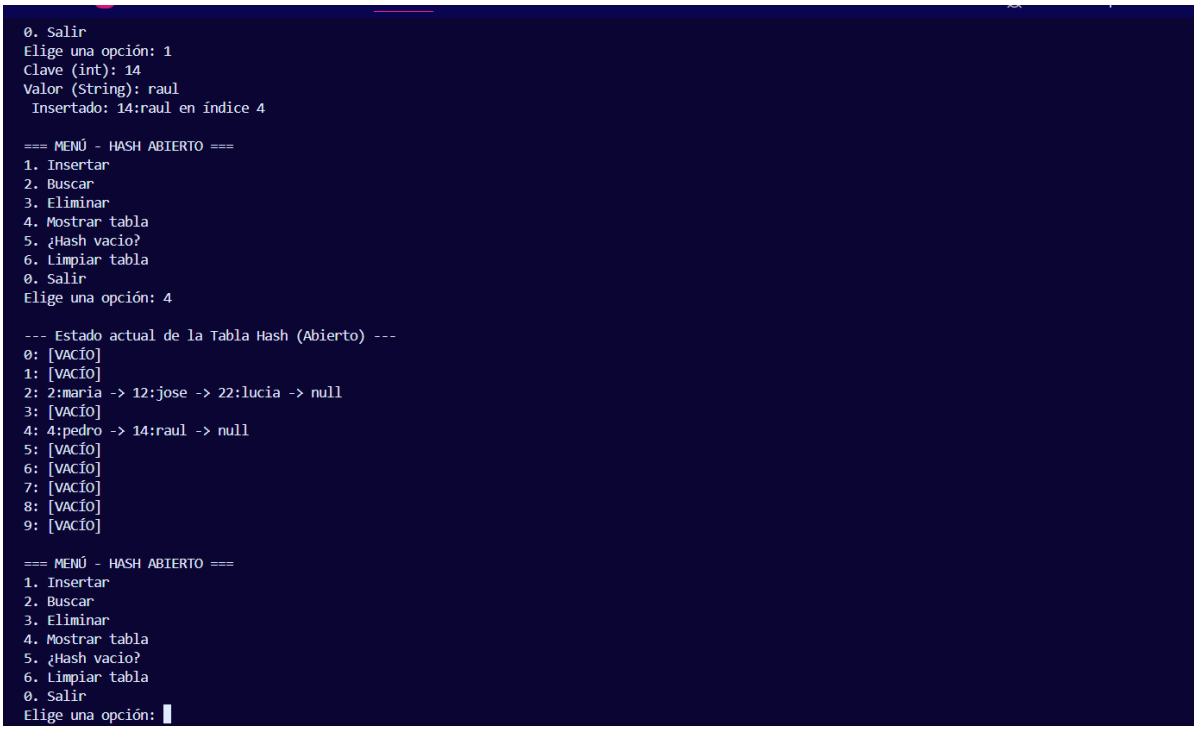
- **HashOpened.java**

```java
    public boolean insert(Register<E> reg) {
        int index = hash(reg.getKey());

        for (Register<E> r : table[index]) {
        if (r.getKey() == reg.getKey() && !r.isDeleted()) {
            System.out.println(" Clave duplicada: " + reg.getKey());
            return false;
        }
        }

        table[index].add(reg);
        System.out.println(" Insertado: " + reg + " en índice " + index);
        return true;
    }

    public boolean delete(int key) {
        int index = hash(key);

        for (Register<E> r : table[index]) {
        if (r.getKey() == key && !r.isDeleted()) {
            r.delete();
            System.out.println(" Eliminado lógicamente: " + key);
            return true;
        }
        }

        System.out.println(" Clave no encontrada: " + key);
        return false;
    }
```

```java
    public Register<E> search(int key) {
        int index = hash(key);

        for (Register<E> r : table[index]) {
        if (r.getKey() == key && !r.isDeleted()) {
            return r;
        }
        }

        return null;
    }

    public boolean isEmpty() {
        for (LinkedList<Register<E>> bucket : table) {
        for (Register<E> r : bucket) {
            if (!r.isDeleted()) return false;
        }
        }
        return true;
    }

    public void clear() {
        for (int i = 0; i < table.length; i++) {
        table[i].clear();
        }
        System.out.println(x:" Tabla vaciada.");
    }

```

```java
    public void showTable() {
        System.out.println(x:"\n--- Estado actual de la Tabla Hash (Abierto) ---");
        for (int i = 0; i < table.length; i++) {
        System.out.print(i + ": ");
        if (table[i].isEmpty()) {
            System.out.println(x:"[VACÍO]");
        } else {
            for (Register<E> r : table[i]) {
            System.out.print(r + " -> ");
            }
            System.out.println(x:"null");
        }
        }
    }
```

- **Resultados**



II. SOLUCIÓN DEL CUESTIONARIO

a. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc

- **Escasa documentación específica:** La mayoría de ejemplos en línea sobre tablas hash en Java no están implementados con clases y métodos genéricos, ni consideran colisiones y eliminación lógica, lo cual dificultó su adaptación.
- **Complejidad del manejo de colisiones:** En el **hash cerrado**, implementar correctamente el **sondeo lineal** para insertar, buscar y eliminar sin errores requería un seguimiento preciso de los índices y validaciones para evitar ciclos o sobreescritura.
- **Gestión de listas enlazadas en hash abierto:** En el **hash abierto**, trabajar con LinkedList<Register<E>> implicó manejar correctamente la inserción, eliminación lógica y la búsqueda dentro de cada lista, cuidando que no se afecten elementos eliminados o duplicados.

- **Prevención de claves duplicadas:** Fue necesario añadir validaciones adicionales para impedir la inserción de claves ya existentes, tanto en el hash cerrado como abierto, lo cual no siempre está presente en implementaciones básicas.
- **Diseño con clases y métodos genéricos:** Hacer que todo el sistema sea totalmente genérico (<E>) implicó mayor atención al tipo de datos y al uso de comparaciones y conversiones, para garantizar la reutilización de las clases con distintos tipos de valores.
- **Implementación del menú de pruebas:** El desarrollo de un menú interactivo que permita probar todas las operaciones de forma clara y controlada requería validaciones constantes y cuidado en la entrada/salida de datos por consola.

III. CONCLUSIONES

- El uso de clases y métodos genéricos permite una mayor reutilización del código, ya que las estructuras implementadas pueden manejar distintos tipos de datos sin necesidad de duplicar la lógica.
- Comprender las técnicas de manejo de colisiones, como el sondeo lineal en el hash cerrado y las listas enlazadas en el hash abierto, es fundamental para garantizar la correcta funcionalidad y eficiencia de una tabla hash.
- La implementación de eliminación lógica es útil para mantener la integridad de la estructura, especialmente cuando se quiere conservar el orden de búsqueda o permitir reutilización de espacios.
- La práctica de validar claves duplicadas al insertar mejora la robustez del programa y evita errores lógicos que podrían comprometer la integridad de los datos almacenados.
- El desarrollo de un menú interactivo de pruebas facilita la verificación y demostración de las operaciones implementadas, y permite detectar posibles errores o mejoras en tiempo de ejecución.
- Finalmente, esta experiencia permite entender en profundidad cómo funcionan internamente estructuras fundamentales en informática como las tablas hash, y su importancia en el desarrollo de sistemas eficientes y escalables.

REFERENCIAS Y BIBLIOGRAFÍA

- Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Estructuras de datos y algoritmos en Java* (6.ª ed.). Pearson Educación.
- Weiss, M. A. (2012). *Data Structures and Algorithm Analysis in Java* (3rd ed.). Pearson.
- Lafore, R. (2003). *Estructuras de datos y algoritmos en Java* (2.ª ed.). Prentice Hall.
- Oracle. (n.d.). *Generics (Updated)*. Oracle Java Documentation. https://docs.oracle.com/javase/tutorial/java/generics/
- Oracle. (n.d.). *Hash Tables*. Java Platform, Standard Edition Documentation. https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html
- GeeksforGeeks. (n.d.). *Hashing in Data Structures*. https://www.geeksforgeeks.org/hashing-data-structure/
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.