	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Tista García Edgar Ing.

Asignatura: Estructura de Datos y Algoritmos II

Grupo: 5

No de Práctica(s): 5

Integrante(s): Téllez González Jorge Luis

No. de Equipo de cómputo empleado: 40

No. de Lista o Brigada: X

Semestre: 2020-2

Fecha de entrega: 11/03/2020

Observaciones:

CALIFICACIÓN: _____

Índice

1. Objetivos

- El estudiante conocerá e identificará las características necesarias para realizar búsquedas por transformación de llaves.
- El alumno conocerá la implementación de la transformación de llaves en el lenguaje orientado a objetos.
- Comprender el manejo básico de los objetos de tipo Hashtable así como sus métodos operativos principales.

2. Entorno de trabajo

A continuación, se ha creado un nuevo proyecto en **NetBeans**. En su clase principal, contiene un menú que permite seleccionar entre 3 operaciones Hash distintas. A su vez, se han creado 3 clases secundarias correspondientes a cada opción disponible.

```
int seleccion = 0;
System.out.println("P5: Algoritmos de busqueda. ");
while (seleccion != 4) {

    System.out.println("¿Que operacion desea utilizar?: ");
    System.out.println("1)Tablas Hash en Java.");
    System.out.println("2)Funcion Hash por modulo.");
    System.out.println("3)Encadenamiento.");
    System.out.println("4)Salir del programa.");
    System.out.println("");

    Scanner menu = new Scanner(System.in);
    seleccion = menu.nextInt();

    switch (seleccion) {
```

Figura 1: *Menú principal del programa.*

Una vez seleccionada una opción, se crea un objeto de tipo *TablaHash*, *FuncionHash* o *Encadenamiento*, dependiendo de la elección, y se accede por medio del mismo al método de acceso a las operaciones de la clase seleccionada.

```
case 1: {

    TablaHash opcionHash= new TablaHash();

    opcionHash.aplicacionTabla();


    break;
```

Figura 2: *Llamada a los objetos específicos para cada opción.*

2.1. Manejo de tablas Hash

La clase *TablaHash* contiene como atributos 3 tablas Hash: una que almacena listas de cadenas y otras 2 tablas que únicamente almacenan cadenas. El primer atributo tiene en consideración una tabla hash que almacene listas de alumnos, considerando la clave como el grado de los alumnos y como valor asociado una lista con sus nombres. Los atributos restantes, en cambio, consideran la clave como el número de cuenta de un alumno y su nombre como valor asociado.

```
public class TablaHash {

    private Hashtable<Integer, ArrayList<String>> alumnos;
    private Hashtable<Integer, String> individuales, equiparables;

    public void aplicacionTabla() {
```

Figura 3: *Atributos de la clase TablasHash.*

A continuación, el programa introduce al usuario a un ejemplo del uso de los métodos de *Hashtable* por medio de las tablas hash que únicamente consideran número de cuenta y nombre del alumno.

```
System.out.println("\n*********Prueba de los métodos de HashTable*********\n");

System.out.println("A continuacion sera creada una Tabla Hash que almacene el nombre "
        + "completo de un alumno y su numero de cuenta como su respectiva clave. \n");

individuales = new Hashtable<>();
```

Figura 4: *Ejemplo inicial.*

- El método **put** agrega en la tabla una clave y su elemento asociado en la tabla:

```
System.out.println("1. Metodo put:");
System.out.println("");

individuales.put(315132726, "Tellez Gonzalez Jorge Luis");
individuales.put(452727153, "Alfonso Perez Manuel");
individuales.put(583738267, "Arteaga Roberto Carlos");
individuales.put(203846379, "Garcia Saavedra Miguel");
individuales.put(310383625, "Gonzalez Noreen Xiadani");
```

Figura 5: *Inserción de un número de cuenta y una cadena asociada.*

```
1. Metodo put:

La Tabla Hash creada contiene los siguientes elementos iniciales...

{452727153=Alfonso Perez Manuel, 315132726=Tellez Gonzalez Jorge Luis, 310383625=Gonzalez Noreen Xiadani, 583738267=Arteaga Roberto Carlos, 203846379=Garcia Saavedra Miguel}
```

Figura 6: *Salida obtenida.*

- El método **contains** permite verificar si un determinado elemento se encuentra en la tabla, introduciéndolo como parámetro.

```
System.out.println("2. Metodo contains:");
System.out.println("¿Existe el alumno Cervantes Moreno Gabriel en la tabla individuales?: "
        + individuales.contains("Cervantes Moreno Gabriel"));
System.out.println("");
```

Figura 7: *Las lineas anteriores verifican si el valor introducido se encuentra presente en la tabla.*

```
2. Metodo contains:
¿Existe el alumno Cervantes Moreno Gabriel en la tabla individuales?: false
```

Figura 8: *Salida obtenida.*

- El método **containsKey** permite verificar si un determinado elemento se encuentra en la tabla, introduciendo su clave asociada.

```
System.out.println("3. Metodo containsKey:");
System.out.println("¿Existe el alumno Téllez González Jorge Luis en la tabla individuales?: "
        + individuales.containsKey(315132726));
System.out.println("");
```

Figura 9: *Las lineas anteriores verifican si la clave introducida se encuentra presente en la tabla.*

```
3. Metodo containsKey:
¿Existe el alumno Téllez González Jorge Luis en la tabla individuales?: true
```

Figura 10: *Salida obtenida.*

- El método **containsValue** actúa de forma idéntica a **contains**.

```
System.out.println("4. Metodo containsValue:");
System.out.println("¿El alumno con cuenta 203846379 se encuentra en la lista?: "
        + individuales.containsValue("Garcia Saavedra Miguel"));
System.out.println("");
```

Figura 11: *Funcionalidad idéntica entre métodos.*

```
4. Metodo containsValue:
¿El alumno con cuenta 203846379 se encuentra en la lista?: true
```

Figura 12: *Salida obtenida.*

- El método **equals** compara 2 tablas Hash, verificando si ambas tablas contienen las mismas claves y elementos asociados.

```
System.out.println("5. Metodo equals:");
equiparables = new Hashtable<>();
equiparables.put(315132726, "Tellez Gonzalez Jorge Luis");
equiparables.put(452727153, "Alfonso Perez Manuel");
equiparables.put(583738267, "Arteaga Roberto Carlos");
equiparables.put(203846379, "Garcia Saavedra Miguel"); //Falta un nombre, la salida resulta false.

System.out.println("¿La Tabla Hash equiparables e individuales contienen las mismas claves y elementos?: "
        + individuales.equals(equiparables));
System.out.println("");
```

Figura 13: *Creación de una segunda tabla hash y uso del método* ***equals***.

```
5. Metodo equals:
¿La Tabla Hash equiparables e individuales contienen las mismas claves y elementos?: false
```

Figura 14: *Salida obtenida.*

- El método **get** recupera el valor almacenado en una clave presente en la tabla Hash especificada.

```
System.out.println("6. Metodo get:");
System.out.println("El Nombre del alumno con cuenta 452727153 es: "
        + individuales.get(452727153));
System.out.println("");
```

Figura 15: *Recuperación del valor asociado a la clave 452727153.*

```
6. Metodo get:
El Nombre del alumno con cuenta 452727153 es: Alfonso Perez Manuel
```

Figura 16: *Salida obtenida.*

- El método **remove** elimina una clave, introducida como parámetro, en la tabla especificada.

```
System.out.println("7. Metodo remove:");
System.out.println("Expulsaron a Alfonso...");
individuales.remove(452727153);
System.out.println("");
```

Figura 17: *Recuperación del valor asociado a la clave 452727153.*

```
7. Metodo remove:
Expulsaron a Alfonso...
```

Figura 18: *Salida obtenida.*

- El método **size** devuelve el número de claves almacenadas en la tabla. Debido a que se eliminó una clave previamente, la salida obtenida es de únicamente 4 claves frente a las 5 iniciales.

```
System.out.println("8. Metodo size:");
System.out.println("¿Cuantos alumnos quedan en la tabla?: " + individuales.size());
System.out.println("Jajajaj eso le pasa por reirse en los honores.");
System.out.println(individuales);
System.out.println("");
```

Figura 19: *Recuperación del número de claves presentes en la tabla individuales.*

```
8. Metodo size:
¿Cuantos alumnos quedan en la tabla?: 4
Jajajaj eso le pasa por reirse en los honores.
{315132726=Tellez Gonzalez Jorge Luis, 310383625=Gonzalez Noreen Xiadani, 583738267=Arteaga Roberto Carlos, 203846379=Garcia Saavedra Miguel}
```

Figura 20: *Salida obtenida.*

Tras la ejecución del ejemplo inicial fue añadido un segundo ejemplo que introduce los mismos conceptos; con la diferencia de que ahora se utiliza una tabla hash que considera la clave como el grado de un grupo de alumnos y como su valor asociado una lista de cadenas con los nombres de cada alumno. Este ejemplo puede omitirse, si el usuario así lo desea.

```
¿Desea observar un ejemplo adicional con listas?
1)Si
2)No

Seleccion: 1
*********Segunda Prueba de los métodos de HashTable*********

A continuacion sera creada una Tabla Hash que almacene listas con nombres de alumnos y su respectivo grado escolar como identificador asociado.

La lista A contiene a los alumnos de primer grado: [Téllez González Jorge Luis, Salazar Fuentes Enrique Antonio]
La lista B contiene a los alumnos de tercer grado: [Fuentes de Ortiz Manuel, Barbosa Mendez Alejandro , Roberto Carlos]
```

Figura 21: *Vista previa del segundo ejemplo.*

2.2. Simulación Hash por módulo

La clase *FuncionHash* contiene 2 atributos: un arreglo dinámico de enteros *ArrayInt* y un escáner *lectura* declarado previamente.

```
public class FuncionHash {

    private ArrayList<Integer> arrayInt;
    public Scanner lectura;
```

Figura 22: *Atributos de la clase FuncionHash.*

Por otra parte, se tienen los siguientes métodos:

- El método **inicializacion** crea un arreglo dinámico de enteros y, por medio de un ciclo **for**, inicializa 15 índices del mismo con un valor **null**.

```
public void inicialización() {

    arrayInt = new ArrayList<>();
    for (int i = 0; i < 15; i++) {

        this.arrayInt.add(null);
```

Figura 23: *Inicialización del arreglo dinámico.*

- El método **moduloHash** imprime el valor introducido como parámetro y su identificador asignado por medio del operador %; el valor considerado es $M = 13$. El método devuelve el identificador o clave a utilizar para el valor recibido.

```
public static int moduloHash(int valorIngresado) {

    System.out.println("El valor ingresado es " + valorIngresado +
            " con identificador: " + (valorIngresado%13));
    int identificador = valorIngresado%13;
    return identificador;
```

Figura 24: *Atributos de la clase FuncionHash.*

- El método **pruebaLinealColision** recibe como parámetro el valor a ingresar en el arreglo dinámico. Inicia obteniendo el identificador asignado al valor que fue recibido haciendo uso del método estático **moduloHash**. Por medio de ciclo **while**, se verifica que en la posición en la que tentativamente será agregado el elemento se encuentre un **null**. En caso de que en la posición tentativa exista un valor diferente a **null**, se incrementa en una unidad la posición tentativa y se realizan 2 verificaciones:

```
while (this.arrayInt.get(posicionAIngresar) != null) {
    posicionAIngresar++;
    System.out.println("Se detectó una colisión. Nuevo identificador: "+posicionAIngresar);
```

Figura 25: *Mensaja de estado: indica la aparición de una colisión entre identificadores.*

  - Si la posición tentativa coinicide con el tamaño de la lista, el índice será establecido en 0: esto indica que el arreglo ha sido recorrido desde la posición tentativa hasta el último índice del mismo. Una vez que la posición se establece en 0, se vuelve a recorrer el arreglo hasta llegar a la posición tentativa original.

  - Si la posición modificada coincide con la posición original, se devuelve un mensaje de error y una posición -1, indicando que no fue posible añadir tal elemento debido a que no hay más espacio disponible.

```
if (posicionAIngresar == this.arrayInt.size()) {
    posicionAIngresar = 0;

}
if (posicionAIngresar==FuncionHash.moduloHash(valorAIngresar)) {
    System.out.println("No hay mas espacio disponible en la lista.");
    return -1;
}
```

Figura 26: *Verificaciones del índice o posición del elemento por agregar.*

El método finaliza retornando el índice o posición válida para agregar el elemento, o -1 en caso de que la lista se encuentre llena.

- La tarea del método **busquedaLineal** consiste en recibir como parámetro el valor a buscar, aplicarle la transformación Hash y obtener el índice donde debería de encontrarse el valor ingresado. Si el arreglo en la posición de búsqueda es diferente de **null**, se ingresa a un ciclo **while** el cuál verifica que, mientras el valor buscado no se encuentre en la posición de búsqueda original, se incremente el índice o posición de búsqueda del valor ingresado.

```
if (this.arrayInt.get(indiceABuscar) != null) {

    while (this.arrayInt.get(indiceABuscar) != valorABuscar) {
        indiceABuscar++;
```

Figura 27: *Condiciones iniciales de búsqueda.*

Al igual que el método anterior, si el incremento del índice supera al tamaño del arreglo, este se reestablece a 0. Finalmente, si el índice de búsqueda modificado vuelve a tomar su valor original, se confirma que el valor buscado no existe en la lista, retornando -1. En caso de que la condición del **while** se rompa en una iteración, el método devuelve el índice de búsqueda donde se encuentra el valor ingresado.

Un índice -1 también será retornado en caso de que el arreglo se encuentre inicializado en todos sus índices con el valor **null**.

```
    if (indiceABuscar == this.arrayInt.size() - 1) {
        indiceABuscar = 0;

    }

    if (indiceABuscar == FuncionHash.moduloHash(valorABuscar)) {
        return -1;
    }

}
return indiceABuscar;
```

Figura 28: *Verificaciones y posibles salidas.*

El método **submenuHash** recibe al objeto creado en la clase principal para dar inicio a la ejecución de la opción seleccionada. Además, se inicializa la lista de enteros previo a la ejecución de cualquier operación usando el método **inicializacion**. Se incorpora un menú que pregunta al usuario qué operación desea realizar:

```
¿Que desea hacer?:
1)Agregar elementos.
2)Imprimir lista.
3)Buscar elementos.
4)Salir de la opcion seleccionada.

Seleccion:
```

Figura 29: *Submenú de FuncionHash.*

- **Agregar elementos:** A continuación se realiza la lectura del valor a ingresar. Posteriormente, se obtiene la posición de inserción utilizando el método **pruebaLinealColision** enviando como parámetro el valor leído. Si la posición obtenida resulta diferente de -1, se inserta en la lista el valor junto con su posición por medio del método **set** y se imprime una confirmación del resultado.

```
Seleccion: 1

Ingrese un valor: 154
El valor ingresado es 154 con identificador: 11

Operacion completada con exito.

[null, null, null, null, null, null, null, null, null, null, null, 154, null, null, null]
```

Figura 30: *Añadidura exitosa de un valor entero a la lista.*

```
El valor ingresado es 2 con identificador: 2
No hay mas espacio disponible en la lista.

El valor no pudo ser agregado.
```

Figura 31: *Mensaje de error obtenido cuando la lista se encuentra llena.*

- **Imprimir lista:** Imprime el estado actual de la lista.

```
Seleccion: 2
Impresion de la lista actual:

[546, 64, 54, 7556, 675, 2345, 863, 64, 643, 243, 7432, 154, 324, 35, 867]
```

Figura 32: *Mensaje de error obtenido cuando la lista se encuentra llena.*

- **Buscar elementos:** Se realiza la lectura del valor a buscar. Luego, se obtiene la posición de búsqueda utilizando el método **busquedaLineal** e ingresando como parámetro el valor leído. Si la posición obtenida es diferente de -1, se imprime en pantalla el índice de coincidencia y la impresión de la lista para verificar la validez de la búsqueda. En caso contrario, se imprime un mensaje de error en pantalla.

```
Seleccion: 3

Ingrese un valor: 862
El valor ingresado es 862 con identificador: 4

El elemento buscado se encuentra en la posicion: 4
[468, 482, 976, 6542, 862, 356, 123, 4687, 346, 875, 369, 236, 896, 285, 467]

Operacion completada con exito.
```

Figura 33: *Operación de búsqueda exitosa.*

```
No se ha encontrado el elemento en la posicion actual. Nuevo identificador: 5
El valor ingresado es 343 con identificador: 5

El valor ingresado no se encuentra en la lista.

[468, 482, 976, 6542, 862, 356, 123, 4687, 346, 875, 369, 236, 896, 285, 467]
```

Figura 34: *Mensaje de error obtenido al introducir en la búsqueda un valor no existente en la lista.*

2.3. Encadenamiento

La clase *Encadenamiento* posee como atributos declarados una tabla hash que almacena claves enteras y arreglos dinámicos como valor asociado, un arreglo dinámico de enteros, un arreglo anidado de listas de enteros y una referencia a un objeto de tipo *Scanner*.

```
private Hashtable<Integer, ArrayList<Integer>> listasEnteros;
private ArrayList<Integer> listaEntero;
private ArrayList<ArrayList<Integer>> listaAEntero;
public Scanner lectura;
```

Figura 35: *Atributos de la clase.*

La clase posee implementados 4 métodos: **inicializacion**, **inicializacionAnidada**, **submenuEncad** y **funcionAleatoria**:

- La inicialización de la tabla Hash consiste en la creación respectiva de la tabla por medio del operador *new* así como en la creación de las respectivas 15 listas por medio de un ciclo **for**; asignando a cada una de ellas una clave del 1 al 15.

```
listasEnteros = new Hashtable<>();

for (int i = 0; i < 15; i++) {

    listasEnteros.put(i, new ArrayList<>());
```

Figura 36: *Hashtable de listas inicializado.*

- Para inicializar la lista anidada de enteros se sigue un procedimiento idéntico; con la única diferencia en el método utilizado para realizar la inserción.

```
listaAEntero = new ArrayList<>();

for (int i = 0; i < 15; i++) {

    listaAEntero.add(i, new ArrayList<>());
```

Figura 37: *Lista de listas inicializada.*

- El método **funcionAleatoria** recibe un entero correspondiente al valor que el usuario desea ingresar. A continuación, se crea un objeto de tipo *Random* indicando como parámetro en el constructor *System.currentTimeMillis()*; de modo que la generación de números aleatorios se basará en la hora actual del sistema.

Hecho lo anterior, se crea una variable entera denominada *posicion*, la cuál almacena el valor que retorne el método **nextInt** aplicado sobre el objeto *posicionAleatoria* de tipo *Random*. Finalmente, se imprime en pantalla un mensaje de estado, indicando el valor ingresado y la lista a la que fue asignado.

```
int posicion = posicionAleatoria.nextInt(15);
System.out.println("El elemento " + valorIngresado +
        " ha sido ingresado a la lista: " + posicion);
return posicion;
```

Figura 38: *Se especifica como parámetro que el valor aleatorio máximo posible sea de 15.*

- El sub-menú del programa permite al usuario elegir entre agregar un elemento a una de las sub-listas, por medio de las 2 implementaciones propuestas, o salir de la sección. Una vez que se lee el valor a insertar, se guarda el valor en una variable entera y se llama al método **funcionRandom** para obtener la posición en la que será almacenado el valor (la posición corresponde a la clave de asociación a cada sub-lista). Finalmente, se imprime el estado actual de la tabla (la lista principal) y sus sublistas.

```
int valorAInsertar = lectura.nextInt();
int claveLista = funcionAleatoria(valorAInsertar);
listasEnteros.get(claveLista).add(valorAInsertar);
```

Figura 39: *Métodos utilizados para ingresar el valor a una de las sub-listas de forma aleatoria..*

```
Seleccion: 3
¿Que desea hacer?:
1)Agregar elementos (Uso de tablas hash).
2)Agregar elementos (Uso de arreglos anidados)
3)Salir de la opcion seleccionada.

Seleccion: 1

Ingrese el número que desea agregar a la lista: 653
El elemento 653 ha sido ingresado a la lista: 3
{14=[], 13=[], 12=[], 11=[], 10=[], 9=[], 8=[], 7=[], 6=[], 5=[], 4=[], 3=[653], 2=[], 1=[], 0=[]}
```

Figura 40: *Impresión de la tabla hash y las sub-listas con sus valores.*

```
Seleccion: 2

Ingrese el número que desea agregar a la lista: 653
El elemento 653 ha sido ingresado a la lista: 4
[[], [], [], [], [653], [], [], [], [], [], [], [], [], []]
```

Figura 41: *Impresión de la lista principal y las sub-listas con sus valores.*

3. Conclusiones

El trabajo realizado con las tablas hash resultó en una experiencia de aprendizaje satisfactoria, ya que su uso abre puertas a diferentes posibilidades de búsqueda con respecto a los algoritmos analizados en la anterior práctica. Pese a que las ventajas más evidentes de su uso se ven mermadas debido a las límitaciones físicas y el compromiso espacio-temporal, su uso resulta práctico e intuitivo.

En términos operativo, puedo mencionar los siguientes puntos:

- El objetivo principal se ha cumplido al obtener una implementación eficaz de una función Hash por módulo capaz de resolver colisiones cuando estas lleguen a presentarse.
- Se obtuvo un aprendizaje notable en el uso de tablas hash y listas anidadas en el entorno orientado a objetos de Java.
- Se ha comprendido de mejor forma el concepto de transformaciones Hash estudiado en la clase teórica y cómo se traduce en una aplicación real de búsqueda de datos.

Una de las principales dificultades se encontró en la delimitación de las restricciones de la prueba lineal durante la aparición de colisiones. Este problema finalmente se resolvió con las condicionales **if** implementadas que permiten garantizar el recorrido de toda la lista en búsqueda de un espacio para colocar el elemento con identificador repetido.

Finalmente, considero que la experiencia obtenida durante la práctica mejorará mis habilidades de análisis y resolución de problemas de forma general. He puesto un fuerte empeño en aprender y trabajar con mayor rapidez, sin embargo, en ocasiones esto resulta complicado por la carga de trabajo presente en otras asignaturas. Son situaciones que, sin embargo, siempre se presentarán en la vida: lidiar con ellas efectivamente y entregar resultados es una tarea obligatoria para cualquier Ingeniero de calidad.