	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación

salas A y B

Profesor : Edgar Tista García

Asignatura: EDA 2

Grupo: 4

No. Práctica : 5

Integrante: Chagoya Gonzalez Leonardo

No. Equipo de Cómputo: N/A

No. de Lista o Brigada: # Lista 08

Semestre: 2022-2

Fecha de entrega: 18 de marzo del 2022

Observaciones:

CALIFICACIÓN: _____

OBJETIVO: El estudiante conocerá e identificará las características necesarias para realizar búsquedas por transformación de llaves

Objetivo de clase: El alumno conocerá la implementación de la transformación de llaves en el lenguaje orientado a objetos

Ejercicio 1

ContainsValue: Recibe como parámetro un valor, el método buscará dentro de la tabla hash si dicho valor existe, en caso de que sea así retorna un true, en caso contrario retorna un False. (Para nuestra práctica los valores son string mientras que las claves son enteros de 9 dígitos).

ContainsKey: El método espera recibir un valor que es la clave de los elementos en la tabla Hash si dicha clave está asociada a un valor devuelve un true, en caso de que la clave no se encuentre devolverá un false.

Equals: Compara si dos objetos son equivalentes en cuanto a sus contenidos y arroja un valor booleano true si la afirmación de equivalencia es cierta y false en caso contrario, para esta práctica se utilizaron 3 tablas hash a comparar.

Get: El método recibe como parámetro un valor que será la clave de un valor asociado en la tabla hash, en caso de que dicha clave este asociada a un valor devuelve un true, en caso contrario devuelve un null.

Put: Este método recibe como parámetro la clave y el valor al que estará asociado dicha clave, posteriormente añade esta relación a la tabla Hash.

Remove: El método remove tiene una sobrecarga, si se le pasa un único parámetro este deberá ser el valor de la clave posteriormente retorna el valor asociado a dicha clave eliminando su relación en la tabla hash, en caso de que se le pasen dos parámetros el primero deberá ser la llave y el segundo el valor al cual se encuentra asociado en caso de que esta relación exista en la tabla Hash se eliminará y devolverá un true, en caso contrario devolverá un false.

Size: Devuelve el número de relaciones clave- valor que tiene nuestra tabla Hash.

Capturas de Ejecución:

Nuestra tabla hash será en un inicio. Podremos llamarle tabla 1

```
La tabla hash es
Llave           Valor
313599345       Ivan Mateos
318845761       Omar Aguirre
318321467       Andrea Rojas
318539822       Vanesa Nava
313667425       Heber Perez
318218814       Leonardo Chagoya
319865833       Angel Serrano
318944924       Alexa Quero
******************
```

Se aplicaron los métodos containsKey y containsValue utilizando llaves y valores que existan dentro de la tabla, así como valores y llaves que no se encontraran.

```
***Utilizando el metodo containsKey***
La clave: 318539822 se encuentra en la tabla: true
La clave 3111234 se encuentra en la tabla: false

***Utilizando el metodo containsValue***
El valor: Ivan Mateos se encuentra asignado por una o mas claves: true
El valor Nikki se encuentra asignado por una o mas claves: false
```

Para aplicar el método equals se creó una tabla hash que solo tuviese un valor y otra tabla hash exactamente igual a tabla 1.

```
****Utilizando el metodo equalss****
Tabla 1
La tabla hash es
Llave           Valor
313599345       Ivan Mateos
318845761       Omar Aguirre
318321467       Andrea Rojas
318539822       Vanesa Nava
313667425       Heber Perez
318218814       Leonardo Chagoya
319865833       Angel Serrano
318944924       Alexa Quero
******************
Tabla 2
La tabla hash es
Llave           Valor
318539822       Ivan Mateos
******************
Tabla 3
La tabla hash es
Llave           Valor
313599345       Ivan Mateos
318845761       Omar Aguirre
318321467       Andrea Rojas
318539822       Vanesa Nava
313667425       Heber Perez
318218814       Leonardo Chagoya
319865833       Angel Serrano
318944924       Alexa Quero
******************
Comparando la igualdad de tabla 1 y tabla 2: false
Comparando la igualdad de tabla 1 y tabla 3: true
```

Se aplicó el método get a *tabla 1* buscando mediante claves los valores asociados a ella. Fueron ingresadas dos claves una que exista dentro de la tabla hash y otra que no exista.

```
****Utilizando el metodo get****
Retornando el valor para la clave 123456789: null
Retornando el valor para la clave 318845761 valor: Omar Aguirre
```

Para el método put se agregó el valor Lemure asociado a la clave 123456789 en *tabla 1* y posteriormente se imprimió dicha tabla.

```
****Utilizando el metodo put****
Agregando a tabla 1 el valor Lemure asociado a la clave 123456789
La tabla hash es
Llave           Valor
313599345       Ivan Mateos
318845761       Omar Aguirre
318321467       Andrea Rojas
318539822       Vanesa Nava
313667425       Heber Perez
318218814       Leonardo Chagoya
319865833       Angel Serrano
318944924       Alexa Quero
123456789       lemure
******************
```

Utilizamos el método remove con sus sobrecargas en tabla 1, ya que para la primera línea solo se le pasa la clave y retorna el valor asociado a dicha clave. Posteriormente se ocupa el mismo método remove solo que se le pasa como parámetros una clave y el valor a dicha clave eliminando la relación y devolviendo un true o si la relación no se encuentra retorna un false

```
****Utilizando el metodo remove***
Eliminando de tabla 1 el valor asociado a la clave 318218814 ,valor removido: Leonardo Chagoya
Eliminando de tabla 1 la clave 666666666 asociado a la clave kira   false
Eliminando de tabla 1 la clave 123456789 asociado a la clave Vanesa Nava   false
Eliminando de tabla 1 la clave 318539822 asociado a la clave Vanesa Nava   true
La tabla hash es
Llave           Valor
313599345       Ivan Mateos
318845761       Omar Aguirre
318321467       Andrea Rojas
313667425       Heber Perez
319865833       Angel Serrano
318944924       Alexa Quero
123456789       lemure
******************
```

Es importante notar que en caso de requerir eliminar una relación clave.-valor esta debe de ser específica ya que en la línea 3 de esta impresión tenemos una clave que existe (123456789) dentro de la tabla hash y un valor que igualmente existe (Vanesa Nava) pero ambos no guardan ninguna relación y es por eso que no se puede eliminar ninguno de los dos elementos -retorno de un false-.

Posteriormente se utiliza el método size indicando cuántas relaciones clave- valor existen dentro de nuestra tabla hash.

```
La tabla hash es
Llave           Valor
313599345       Ivan Mateos
318845761       Omar Aguirre
318321467       Andrea Rojas
313667425       Heber Perez
319865833       Angel Serrano
318944924       Alexa Quero
123456789       lemure
******************

****Utilizando el metodo size*****
El numero de asignaciones clave-valor en tabla 1 es: 7
```

Este ejercicio se llevó a cabo de manera exitosa y sin mayor problema pues únicamente se investigó de forma general la clase HashMap y probaron algunos de sus métodos, permitiéndonos observar la implementación de las tablas Hash en Java. La informacion fue recuperada de la documentacion ubicada en la direccion: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

Ejercicio 2:

Para este ejercicio dentro de la clase *HashModulo* se desarrolló un método llamado *menú* dentro de este se instancia una lista la cual tendrá 15 elementos todos inicializados en null. Posteriormente se creó el menú de opciones para agregar, imprimir y buscar por medio del valor.

Para el método de agregar es donde se presentó mayor dificultad pues se tiene que leer un número de 9 dígitos , para verificar que el número sea correcto se realizó un método que nos indicará si el numero ingresado es de nueve dígitos en caso contrario seguirá pidiendo al usuario un número nuevo , posteriormente se tiene que aplicar el plegamiento a dicho valor el resultado de plegamiento indicará la posición en la lista en que se encontrará inicialmente el valor.

Para realizar plegamiento se divide el valor en bloques de 4, se suman los bloques y a los dos últimos dígitos de la suma de los bloques se le aplica la operación módulo 15 , esta parte del algoritmo fue resuelta mediante divisiones y módulos dentro de el método Plegamiento que retornara el valor de las operaciones correspondientes.

Es importante mencionar que se utilizó una clave auxiliar pues para obtener los bloques de esta forma se tenía que modificar el valor ingresado y como dicho valor se utiliza para agregarlo a la lista este no podía modificarse.

```
//obtencion del primero bloque de 4 digitos
bloque = claveAux / 100000;
claveAux = claveAux % 100000;
System.out.println("El bloque de informacion es: "+bloque);
sumaBloques += bloque;

//obtencion del segundo bloque de 4 digitos
bloque = claveAux / 10;
System.out.println("El bloque de informacion es: "+bloque);
sumaBloques += bloque;

//obtencion del ultimo digito
bloque = claveAux % 10;
System.out.println("El bloque de informacion es: "+bloque);
sumaBloques += bloque;

System.out.println("La suma de los bloques es: "+sumaBloques);

bloque = sumaBloques %100; //obtencion de los 2 ultimos digitos
System.out.println("Los dos ultimos digitos: "+bloque);
pos = bloque%15;
System.out.println("La posicion inicial del elemento debe de ser: "+pos);
return pos;
```

Si el valor de estas operaciones retorna una posición en la cual el valor de la lista sea igual a null se inserta el valor en dicha posición, pero si se retorna una posición en la cual ya se encuentra un elemento - la posición en la lista tiene un elemento diferente de null- se recorre lo que queda de la lista hasta encontrar una posición

disponible e inserta el número. Si ya no hay más posiciones disponibles hacia delante no se inserta el número.

Aquí no se utiliza el método add sobrecargado indicando la posición y el valor a colocar pues este método generará más elementos- si mi lista es de 15 y se añade un elemento la lista será de 16 elementos- en cambio se utiliza set(j,valor) ya que este sobrescribe la información *valor* en la posición indicada por *j.*

EJECUCIÓN

```
*******FUNCION HASH POR MODULO********
Lista inicializada con 15 elementos nulos

------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 1

******AGREGAR*****
Ingrese el valor:
123456789
El valor ingresado es correcto
El bloque de informacion es: 1234
El bloque de informacion es: 5678
El bloque de informacion es: 9
La suma de los bloques es: 6921
Los dos ultimos digitos: 21
La posicion inicial del elemento debe de ser: 6
El valor 123456789 fue ingresado en la posicion 6
```

```
------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 1

******AGREGAR*****
Ingrese el valor:
987654321
El valor ingresado es correcto
El bloque de informacion es: 9876
El bloque de informacion es: 5432
El bloque de informacion es: 1
La suma de los bloques es: 15309
Los dos ultimos digitos: 9
La posicion inicial del elemento debe de ser: 9
El valor 987654321 fue ingresado en la posicion 9
```

```
------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 1

******AGREGAR*****
Ingrese el valor:
318218814
El valor ingresado es correcto
El bloque de informacion es: 3182
El bloque de informacion es: 1881
El bloque de informacion es: 4
La suma de los bloques es: 5067
Los dos ultimos digitos: 67
La posicion inicial del elemento debe de ser: 7
El valor 318218814 fue ingresado en la posicion 7
```

```
------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 1

******AGREGAR*****
Ingrese el valor:
490917853
El valor ingresado es correcto
El bloque de informacion es: 4909
El bloque de informacion es: 1785
El bloque de informacion es: 3
La suma de los bloques es: 6697
Los dos ultimos digitos: 97
La posicion inicial del elemento debe de ser: 7

Hay una colision la posicion esta siendo ocupada!!

El valor 490917853 fue ingresado en la posicion 8
```

```
------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 1

******AGREGAR*****
Ingrese el valor:
318218814
El valor ingresado es correcto
El bloque de informacion es: 3182
El bloque de informacion es: 1881
El bloque de informacion es: 4
La suma de los bloques es: 5067
Los dos ultimos digitos: 67
La posicion inicial del elemento debe de ser: 7

Hay una colision la posicion esta siendo ocupada!!

El valor 318218814 fue ingresado en la posicion 10
```

```
------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 1

******AGREGAR*****
Ingrese el valor:
111111111
El valor ingresado es correcto
El bloque de informacion es: 1111
El bloque de informacion es: 1111
El bloque de informacion es: 1
La suma de los bloques es: 2223
Los dos ultimos digitos: 23
La posicion inicial del elemento debe de ser: 8

Hay una colision la posicion esta siendo ocupada!!

El valor 111111111 fue ingresado en la posicion 11
```

Para la impresión se creó el método Imprimir el cual recibe como parámetro la lista que se desea imprimir y realiza dicha operación mediante un ciclo for each aquí en caso de que un elemento de la lista no contenga ningún valor imprimirá null

```
------------MENU------------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 2

La lista al momento es:
Llave/Posicion  Valor
0               null
1               null
2               null
3               null
4               null
5               null
6               123456789
7               318218814
8               490917853
9               987654321
10              318218814
11              111111111
12              null
13              null
14              null
```

Para la búsqueda es un pensamiento similar a ingresar ya que se aplica la operación plegamiento en el valor a buscar, se retorna una posición y si en dicha posición no se encuentra el elemento pero aún hay más posiciones en la lista se recorre hasta encontrar el valor o llegar el final de la lista.

Buscando un valor que se encuentra en la lista

```
-----------MENU-----------
1.-Agregar Elementos
2.-Imprimir Lista
3.-Buscar Elementos
4.-Salir
Opcion: 3

******BUSQUEDA*****
Ingrese el valor:
318218814
El valor ingresado es correcto
El bloque de informacion es: 3182
El bloque de informacion es: 1881
El bloque de informacion es: 4
La suma de los bloques es: 5067
Los dos ultimos digitos: 67
La posicion inicial del elemento debe de ser: 7
El valor 318218814 fue encontrado en la posicion 7
```

Buscando un valor que no se encuentra en la lista

```
******BUSQUEDA*****
Ingrese el valor:
987654321
El valor ingresado es correcto
El bloque de informacion es: 9876
El bloque de informacion es: 5432
El bloque de informacion es: 1
La suma de los bloques es: 15309
Los dos ultimos digitos: 9
La posicion inicial del elemento debe de ser: 9
El valor 987654321 no fue encontrado en la lista
```

Ejercicio 3:

Para este ejercicio la mayor dificultad se presentó en comprender cómo instanciar un objeto que sea una lista y que en cada uno de los elementos sea una lista, en otras palabras se debía de crear una lista de listas.

Para realizar esto se instancia un objeto LinkedList que podrá ingresar elementos del tipo Lista que a su vez esta lista tendrá elementos numéricos.

Esto solo instancia la primera lista pero en cada uno de los elementos se debe instanciar las listas correspondientes, el ejercicio solicita que se cree una lista de 15 listas de enteros esto se realiza dentro del ciclo for donde para cada elemento de nuestra lista se añade una lista de enteros.

```
List<List<Integer>> listaDeListas = new LinkedList<List<Integer>>();
for (int i = 0; i < 15; i++) {
    listaDeListas.add(new LinkedList<Integer>()); //geenerar dentro c
}
```

Posteriormente si el usuario desea agregar un elemento ingresará dicho elemento y nuestra función random arroja un valor numérico al azar entre 0 y 14, accedemos desde nuestra lista de listas al elemento en la posición indicado por clave ,que será una lista, una vez ahí añadiremos el valor ingresado y se imprimirá nuestra lista de listas. Aquí las colisiones se resuelven de forma automática ya que nuestros elementos son listas y cada vez que se accede a un elemento lista se agrega el valor al final de esta dando un mayor dinamismo al programa.

```
listaDeListas.get(clave).add(valor); //Agregar elemento por encadenamiento
```

```
********ENCADENAMIENTO********
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 30
EL valor 30 fue colocado en la poscion 3
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2:
LISTA 3: 30
LISTA 4:
LISTA 5:
LISTA 6:
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10:
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14:
```

```
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 4
EL valor 4 fue colocado en la poscion 14
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2:
LISTA 3: 30
LISTA 4:
LISTA 5:
LISTA 6:
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10:
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14: 4
```

```
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 17
EL valor 17 fue colocado en la poscion 14
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2:
LISTA 3: 30
LISTA 4:
LISTA 5:
LISTA 6:
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10:
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14: 4 17
```

```
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 1
EL valor 1 fue colocado en la poscion 2
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2: 1
LISTA 3: 30
LISTA 4:
LISTA 5:
LISTA 6:
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10:
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14: 4 17
```

```
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 10
EL valor 10 fue colocado en la poscion 5
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2: 1
LISTA 3: 30
LISTA 4:
LISTA 5: 10
LISTA 6:
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10:
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14: 4 17
```

```
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 40
EL valor 40 fue colocado en la poscion 6
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2: 1
LISTA 3: 30
LISTA 4:
LISTA 5: 10
LISTA 6: 40
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10: 2
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14: 4 17
```

```
------------MENU----------
1.-Agregar Elemento
2.-Salir
Opcion: 1
Ingrese el numero: 777
EL valor 777 fue colocado en la poscion 6
El estado global de la lista es:
LISTA 0:
LISTA 1:
LISTA 2: 1
LISTA 3: 30
LISTA 4:
LISTA 5: 10
LISTA 6: 40 777
LISTA 7:
LISTA 8:
LISTA 9:
LISTA 10: 2
LISTA 11:
LISTA 12:
LISTA 13:
LISTA 14: 4 17
```

Hay que notar que en caso de que la posición para insertar un valor en una lista se repita este es ingresado sin mayor problema como un nodo al final de la lista correspondiente, esto nos permite observar el porqué la solución de colisión por encadenamiento es de los mejores métodos.

Conclusiones

Esta práctica se realizó de forma satisfactoria ya que se completaron cada uno de los ejercicios de acuerdo a lo solicitado, de igual forma se cumplieron los objetivos de la práctica pues observamos y codificamos la búsqueda por transformación de llaves, ya sea apoyándonos de las clases proporcionadas por el lenguaje Java o codificando nosotros mismo los algoritmos.

En el primer ejercicio nos apoyamos de la clase HashMap la cual es una clase que se encuentra definida en Java con cada uno de los métodos correspondientes permitiéndonos observar el funcionamiento de las tablas Hash y la relación que guarda las claves y los valores ingresados.

En el ejercicio 2 fue donde en lo personal se presentaron las mayores dificultades ya que codificamos la teoría de transformación por plegamiento haciendo pliegues de 4 dígitos, manejo de colisiones por prueba lineal y búsqueda estas dos últimas se vieron apoyadas de la transformación por plegamiento haciendo de este método fundamental para el desarrollo del ejercicio.

Para el ejercicio 3 la dificultad se presentó en generar la lista de listas y mediante ella pudimos observar de mejor forma la solución por encadenamiento ya que solo se agregan a una lista ligada los elementos que van en la misma posición si bien es de los mejores métodos para resolver las colisiones esta implica un uso de memoria adicional.

Al final la teoría vista en clase se llevó a la implementación mediante la realización de esta práctica teniendo como mayor dificultad el razonamiento al codificar ciertas partes de los ejercicios,además la práctica al final te va indicando lo que solicita y en este aspecto es clara lo cual facilita la realización de los ejercicios.