	Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia	

Laboratorios de computación salas A y B

Profesor: M.I. EDGAR TISTA GARCÍA

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS 2

Grupo: 8

No de Práctica(s): 5] ALGORITMOS DE BUSQUEDA PARTE 2

Integrante(s): ADOLFO ROMÁN JIMENEZ

No. de Equipo de cómputo empleado: TRABAJO EN CASA

No. de Lista o Brigada:

Semestre: 2022 - 1

Fecha de entrega: 18 DE OCTUBRE DE 2021

CALIFICACIÓN: _____

Practica 5

Algoritmos de Busqueda Parte 2

Adolfo Roman Jimenez

October 18, 2021

1 Objetivo

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

2 Objetivo de Clase

El alumno conocerá la implementación de la transformación de llaves en el lenguaje orientado a objetos.

3 Ejercicio 0: Menu para la practica

3.1 Desarrollo

Para el primer ejercicio el desarrollo fue muy sencillo. Como la practica lo indica, primeramente cree el proyecto en *Java* con el nombre de **Practica5[RomanAdolfo]** y despues procedi a crear la clase principal del proyecto que tambien lleva por nombre *Practica5*.

Esta clase es la unica que contiene a la funcion principal, mientras que todas las demas clases no contienen ninguna.

A continuacion, como la practica lo indicaba, procedi a crear el menu del usuario con las opciones requeridas y una adicional para salir del programa.

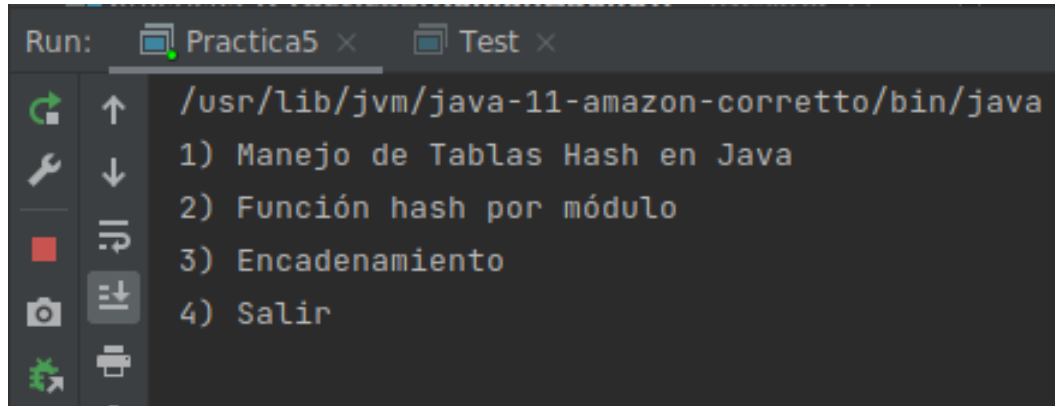
1. Manejo de Tablas Hash en Java
2. Función hash por módulo
3. Encadenamiento
4. Salir

Al principio de la clase se declara `import java.util.Scanner;` para poder hacer uso del *Scanner* posteriormente. Ya dentro de la funcion principal se crea una nueva instancia **scan** del *Scanner* y una variable entera **option**, que almacenara la opcion deseada para navegar en el menu.

Por dentro el menu se compone de dos *do-while* loops indexados, que rompen el ciclo, el primero cuando se escoje la opcion indicada dentro del menu y el segundo, cuando se escoje la opcion numero 4 que es la que permite al usuario terminar el programa.

Despues de que el usuario escoje alguna opcion valida, entre a una estructura *switch* que contiene 3 casos entre los que se encuentra el codigo de los ejercicios que se fueron creando durante la practica.

3.2 Ejecucion



```
Run: Practica5 x Test x
/usr/lib/jvm/java-11-amazon-corretto/bin/java
1) Manejo de Tablas Hash en Java
2) Función hash por módulo
3) Encadenamiento
4) Salir
```

Ejercicio 0: Menu Principal

4 Ejercicio 1: Tablas Hash en Java

4.1 Desarrollo

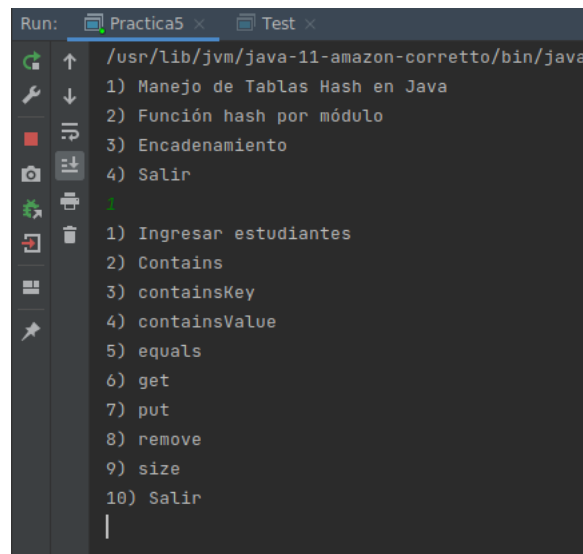
Cuando el usuario ingresa la opcion numero 1 en el menu principal, la estructura *switch* lo llevara al codigo escrito para el ejercicio de las tablas hash.

Para este ejercicio se creo una nueva clase llamada *HashTable* que al principio del codigo, declara importa la clase `import java.util.HashMap;`. Esta clase se usa para crear una tabla hash por medio de la clase *HashMap* para la cual en un inicio, le asignamos un dato de tipo entero como *key* y otro de tipo *string* como *value* y a la nueva instancia de la *HashMap* le ponemos el nombre de **stdlist**.

La clase *HashTable* tambien contiene un menu interno que esta declarado en forma de metodo de la clase llamado *hashMenu()*.

Cuando el usuario en el menu principal escoje la opcion numero 1, la estructura *switch* en el caso uno, primero crea una nueva instancia de la clase *HashTable* a la cual le asigna el nombre de **stdHash** y acto seguido, penetra en la instancia a traves del metodo *hashMenu()* que interactua con todos los metodos internos de la clase para llevar a cabo su proposito.

El menu de el metodo *hashMenu()* es el siguiente:



```
Run: Practica5 x Test x
/usr/lib/jvm/java-11-amazon-corretto/bin/java
1) Manejo de Tablas Hash en Java
2) Función hash por módulo
3) Encadenamiento
4) Salir
5
1) Ingresar estudiantes
2) Contains
3) containsKey
4) containsValue
5) equals
6) get
7) put
8) remove
9) size
10) Salir
|
```

Ejercicio 1: Menu Principal

Explicaremos cada uno de estos metodos y que es lo que hacen:

4.1.1 Ingresar Estudiantes

Cuando en este menu, el usuario elige la opcion numero 1, el programa lo mandara al metodo privado *insertStudents()*. Dentro de este metodo, primeramente y a traves de una leyenda se le solicita al usuario que ingrese la cantidad de estudiantes que desea agregar; para capturar esta informacion se declara la variable entera **number** y se escanea el numero que el usuario desee ingresar.

Acto seguido, a traves de un *while-loop* que iterara la cantidad de veces que el usuario en **number** haya declarado, se invocara al metodo *insertStd()*.

El metodo *insertStd()*, tambien privado, en el momento en el que el usuario ingresa a el, se le pregunta por medio de una impresion en pantalla que inserte el nombre del alumno. A traves de una variable global de tipo *String* llamada **name** y con ayuda del metodo *useDelimiter()* en la instancia *scan()* que permite incluir espacios dentro del *input* del usuario, se deposita la informacion en **name**.

Posteriormente, el programa tambien pide que se ingrese el numero de cuenta del usuario y este se deposita en la variable entera **key**.

Cuando se ha terminado de ingresar los dos valores, a traves del metodo *stdlist.put()* se ingresan los parametros **key** en primer lugar y **name** en segundo lugar que corresponden a la llave y al valor en la tabla hash.

En el ejercicio, esto lo llevamos a cabo diferentes veces con los siguientes nombres:

```
1
No. de estudiantes a ingresar:
4
Inserte nombre:
Adolfo Roman
Inserte cuenta:
410098363
Inserte nombre:
Arturo Herrera
Inserte cuenta:
487541548
Inserte nombre:
Daniela Lopez
Inserte cuenta:
547845126
Inserte nombre:
Alejandra Martinez
Inserte cuenta:
987456215
```

Ejercicio 1: Captura de nombres

4.1.2 Contains

Una vez que los nombres han sido capturados, pasamos a la opción número 2 que corresponde con el método *contains()*.

En este método, a través de la función *isEmpty()* se evalúa que la tabla no esté vacía, si no lo está.

4.2 Ejecucion

```
2
Ingrese numero de cuenta del estudiante:
547845126
Ingrese nombre o apellido del estudiante a buscar:
Martinez
El nombre o apellido del estudiante NO corresponde al numero de cuenta
```

Ejercicio 1: Opcion 2

```
2
Ingrese numero de cuenta del estudiante:
410098363
Ingrese nombre o apellido del estudiante a buscar:
Fernando
El nombre o apellido del estudiante NO corresponde al numero de cuenta
```

Ejercicio 1: Opcion 2

```
3
Ingrese llave a buscar:
410098363
La llave 410098363se encuentra en la lista
```

Ejercicio 1: Opcion 3

```
3
Ingrese llave a buscar:
847451987
La llave 847451987 NO se encuentra en la lista
```

Ejercicio 1: Opcion 3


```
4
Ingrese valor a buscar:
Alejandra Martinez
El nombre Alejandra Martinezse encuentra en la lista
```

Ejercicio 1: Opcion 4

```
4
Ingrese valor a buscar:
Cristobal Hernandez
El nombre Cristobal Hernandez NO se encuentra en la lista
```

Ejercicio 1: Opcion 4

```
6
Ingrese cuenta de estudiante a buscar:
410098363
Nombre asociado a cuenta: Adolfo Roman
```

Ejercicio 1: Opcion 6

```
6
Ingrese cuenta de estudiante a buscar:
874515879
No existe nombre asociado a cuenta
```

Ejercicio 1: Opcion 6

```
7
Ingrese nuevo elemento a tabla Hash:
Inserte nombre:
Cristobal Hernandez
Inserte cuenta:
874515879
```

Ejercicio 1: Opcion 7, ingresando nuevo elemento

```
4
Ingrese valor a buscar:
Cristobal Hernandez
El nombre Cristobal Hernandezse encuentra en la lista
```

Ejercicio 1: Opcion 4, buscando nuevo elemento

```
6
Ingrese cuenta de estudiante a buscar:
874515879
Nombre asociado a cuenta: Cristobal Hernandez
```

Ejercicio 1: Opcion 6, buscando nuevo elemento

```
8
Inserte cuenta de estudiante a remover:
410098363
Estudiante removido
```

Ejercicio 1: Opcion 8

```
6
Ingrese cuenta de estudiante a buscar:
410098363
No existe nombre asociado a cuenta
```

Ejercicio 1: Opcion 6, buscando elemento removido

```
9
La cantidad de elementos en tabla es de 4
```

Ejercicio 1: Opcion 9

```
8
Inserte cuenta de estudiante a remover:
847845154
```

Ejercicio 1: Opcion 8, removiendo elementos

```
8
Inserte cuenta de estudiante a remover:
487541548
```

Ejercicio 1: Opcion 8, removiendo elementos

```
8
Inserte cuenta de estudiante a remover:
547845126
```

Ejercicio 1: Opcion 8, removiendo elementos

```
8
Inserte cuenta de estudiante a remover:
987456215
```

Ejercicio 1: Opcion 8, removiendo elementos

```
9
No existen elementos en tabla
```

Ejercicio 1: Opcion 9, no existen elementos

```
1) Ingresar estudiantes
2) Contains
3) containsKey
4) containsValue
5) equals
6) get
7) put
8) remove
9) size
10) Salir
10
1) Manejo de Tablas Hash en Java
2) Función hash por módulo
3) Encadenamiento
4) Salir
```

Ejercicio 1: Opcion 10, salida del programa

5 Ejercicio 2

5.1 Desarrollo

Cuando el usuario en el menu principal escoge la opcion numero 2, entonces se dirigira al codigo escrito dentro de la clase *HashModulo*.

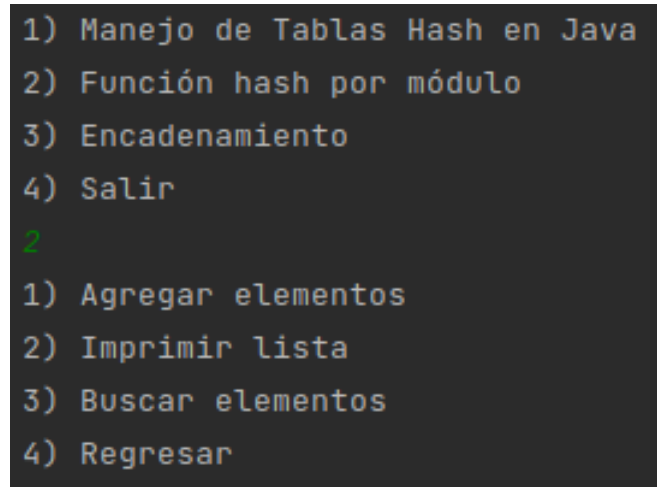
La clase *HashModulo* es una clase cuyo objetivo es demostrar la forma en la que se puede usar una funcion *hash* para almacenar elementos en un arreglo regular a traves de una funcion sencilla que permita encontrar a estos elementos rapidamente cada vez que se les busque.

Para lograr esto, primeramente la clase usa el codigo `import java.util.LinkedList;` para poder hacer uso de las listas ligadas para el funcionamiento del programa.

Cuando ingresamos a la clase, lo primero que haremos sera crear una lista ligada de variables enteras a la cual le llamaremos **lst**.

El constructor de esta clase, que lleva el mismo nombre, carece de parametros iniciales, pero dentro del cuerpo de su metodo yace un *for-loop* que itera 20 veces como el ejercicio lo requiere y agrega unicamente *null* a cada uno de los indices del arreglo.

El menu contiene los elementos requeridos por el ejercicio, que son las acciones de **Ingresar elementos**, **Imprimir lista** y **Buscar elementos** como se muestra en la siguiente imagen.



```
1) Manejo de Tablas Hash en Java
2) Función hash por módulo
3) Encadenamiento
4) Salir
2
1) Agregar elementos
2) Imprimir lista
3) Buscar elementos
4) Regresar
```

Ejercicio 2: Menu Principal

Dependiendo de la opcion que el usuario elija, lo llevara a la opcion correspondiente.

Para la opcion numero 1, lo dirigimos al metodo *addElement()*.

El metodo *addElement()*, utiliza algunas funciones auxiliares. Primeramente solicita al usuario que ingrese el elemento deseado y este se deposita en la variable entera **val**, despues de esto, como sabemos que **val** debe de tener un numero entero de 9 digitos que representa un numero de cuenta, entonces procedemos a obtener la suma de sus digitos a traves del metodo *folding()*.

Dentro del metodo *folding()* que es un metodo que devuelve un entero, declaramos una variable entera **sum** y acarreamos como argumento el entero **val** que contiene al numero de cuenta.

Para poder hacer uso de la tecnica de *folding*, procedemos a lo siguiente.

- Depositamos en **sum** el modulo 10 de var
- Dividimos sobre 10 a **var** y reasignamos su propio valor
- Con un *for-loop* obtenemos el modulo de **var** sobre 10,000 y lo sumamos a **sum**
- Dividimos a **var** sobre 10,000 y reasignamos su propio valor
- Repetimos este proceso 2 veces
- Regresamos el valor en **sum**

A traves de este proceso, el numero de cuenta se divide en 2 conjuntos de 4 digitos y 1 de 1, los cuales se suman para obtener el valor en **sum** que despues se transfiere de igual forma.

Al momento de recibir el resultado, el metodo *addElement()* procede a invocar al metodo *probing()* que incluye a los argumentos **var** que contiene el numero de cuenta y **sum** que contiene el resultado del *folding* del numero de cuenta.

Para *probing()* el proceso es un poco mas sencillo ya que simplemente se hace uso de la formula para *probing* que es la siguiente:

$$\begin{aligned}h'(x) &= (h(x) + f(i))\%n \\ h(x) &= x\%n \\ f(i) &= 0, 1, 2, 3...\end{aligned}$$

Cuando nosotros usamos el *for-loop* para ingresar los elementos a la lista ligada, asignamos una variable de tipo entero **index** dentro de la cual, depositamos el modulo 20 de la variable **sum** mas el valor en **i** del iterador, de nuevo entre el modulo 20.

Esto constantemente nos proporciona indices que se encuentran dentro del arreglo y continua su ejecucion hasta que el metodo es capaz de colocar un elemento dentro del arreglo.

Cuando existe colision, esto es, cuando el indice dentro del que se quiere ingresar el **val** no es igual a *null*, entonces continua el *for-loop* hasta un maximo de 20 veces, pues es la cantidad de espacios disponibles.

El *for-loop* se rompe, en el momento en que un valor logra colocarse dentro del arreglo y este imprime una leyenda indicando el valor colocado y el indice.

5.2 Ejecucion

```
1
Ingrese elemento:
977156731
Elemento: 977156731 almacenado en indice: 5
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
844693419
Elemento: 844693419 almacenado en indice: 16
```

Ejercicio 2: Colocando elementos dentro del arreglo


```
1
Ingrese elemento:
821484488
Elemento: 821484488 almacenado en indice: 10
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
461272674
Elemento: 461272674 almacenado en indice: 3
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
176564592
Elemento: 176564592 almacenado en indice: 6
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
847925566
Elemento: 847925566 almacenado en indice: 1
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
776987219
Elemento: 776987219 almacenado en indice: 19
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
984587734
Elemento: 984587734 almacenado en indice: 2
```

Ejercicio 2: Colocando elementos dentro del arreglo

```
1
Ingrese elemento:
362234617
Elemento: 362234617 almacenado en indice: 11
```

Ejercicio 2: Colocando elementos dentro del arreglo

2

```
Indice: 0, Valor: null
Indice: 1, Valor: 847925566
Indice: 2, Valor: 984587734
Indice: 3, Valor: 461272674
Indice: 4, Valor: null
Indice: 5, Valor: 977156731
Indice: 6, Valor: 176564592
Indice: 7, Valor: null
Indice: 8, Valor: null
Indice: 9, Valor: null
Indice: 10, Valor: 821484488
Indice: 11, Valor: 362234617
Indice: 12, Valor: null
Indice: 13, Valor: null
Indice: 14, Valor: null
Indice: 15, Valor: 985838298
Indice: 16, Valor: 844693419
Indice: 17, Valor: null
Indice: 18, Valor: null
Indice: 19, Valor: 776987219
```

Ejercicio 2: Impresion de elementos dentro de la lista

```
1
Ingrese elemento:
977156731
Elemento: 977156731 almacenado en indice: 7
```

Ejercicio 2: Colocando elemento repetido dentro del arreglo

```
1
Ingrese elemento:
977156731
Elemento: 977156731 almacenado en indice: 8
```

Ejercicio 2: Ingresando elemento repetido

```
3
Ingrese elemento a buscar:
985838298
Elemento: 985838298 se encuentra en indice: 15
```

Ejercicio 2: Buscando elementos dentro del arreglo

```
3
Ingrese elemento a buscar:
176564592
Elemento: 176564592 se encuentra en indice: 6
```

Ejercicio 2: Buscando elementos dentro del arreglo

3

Ingrese elemento a buscar:

977156731

Elemento: 977156731 se encuentra en indice: 5

Elemento: 977156731 se encuentra en indice: 7

Elemento: 977156731 se encuentra en indice: 8

Ejercicio 2: Buscando elemento repetido dentro de arreglo

1) Agregar elementos

2) Imprimir lista

3) Buscar elementos

4) Regresar

4

1) Manejo de Tablas Hash en Java

2) Función hash por módulo

3) Encadenamiento

4) Salir

Ejercicio 2: Salida del programa

6 Ejercicio 3

6.1 Desarrollo

Por ultimo, para cuando el usuario en el menu principal ingresa la opcion numero 3, a traves del *switch* el usuario ingresara al caso 3 que corresponde al codigo de la clase *Encadenamiento*.

Como en las ocasiones anteriores, primero se crea una nueva instancia de la clase llamada **lstEncad** y despues se ingresa en uno de los metodos de esa instancia que tambien lleva al menu principal a traves del metodo *menuEncadenamiento()*.

La clase *Encadenamiento* corresponde al ejercicio de "lista de listas" y es por esta razon que al principio de la clase se declaran distintas clases para importar como lo son:

- `import java.util.ArrayList;`
- `import java.util.List;`
- `import java.util.Random;`

Las primera *ArrayList* nos servira para la lista de enteros que queremos crear, mientras que la segunda *List* nos funcionara para hacer la lista principal que contendra a las demas listas.

La clase *Random* nos servira para poder realizar lo que el ejercicio pide, ya que con ella generaremos numeros al azar para poder ingresar elementos en posiciones al azar como el ejercicio lo marca.

Una vez que hemos ingresado a la clase, lo primero que encontramos es la declaracion de una *List* cuyos elementos son del tipo *ArrayListInteger* que es una sublista que contiene variables enteras. A esta lista principal le llamamos **masterList**.

Despues de crear las instancias requeridas tanto de *Random* como de *Scanner* para su uso posterior, pasamos al constructor de la clase *Encadenamiento*.

El constructor de la clase es uno que no tiene parametros, pero dentro de su cuerpo encontramos un *for-loop* que itera 15 veces de acuerdo a lo requerido por el ejercicio y cada vez que lo hace, creara una lista de enteros *ArrayListi* que tendra como nombre **sublista** y acto seguido a traves del metodo *add()* en **masterList** agregara esa lista creada por cada iteracion.

Cabe mencionar que a pesar de que la lista generada **sublista** siempre tiene el mismo nombre, por cada nueva instancia creada, el programa en realidad crea un objeto distinto dentro de la memoria con un numero de referencia distinto que apunta a otro objeto y no al mismo. Caso contrario, si hubieramos creado una instancia *antes* de ingresar al *for-loop* y esa misma instancia de lista la hubieramos ingresado 15 veces a **masterList** entonces simplemente habriamos ingresado 15 veces la misma referencia al mismo objeto, por lo cual cada elemento agregado en esa referencia, seria el mismo para "todas las demas" pues

la referencia seria la misma.

Esto solo es el constructor, para el menu, como el ejercicio lo solicita, unicamente agregamos 2 opciones que son:

1. Agregar elementos
2. Salir

Dependiendo de la opcion que el usuario elija, sera lo que el programa realice, pero si escoge la opcion 1, entonces el programa lo mandara hacia el metodo *allTheStuffThisDoes()* que contiene un par de metodos adicionales los cuales son *addElement()* y *print()*.

El metodo *addElement()* primeramente solicita al usuario textualmente el elemento a agregar.

Por medio de una variable entera **element** se captura el elemento que el usuario desee agregar. Despues de esto, se declara la variable entera **index** la cual se auxilia de la instancia de la clase *Random* para formular un numero al azar entre 0 y 14, lo que corresponde a la cantidad de indices disponibles dentro de la **masterList**.

Acto seguido, se usan dos metodos en **masterList** para ingresar el elemento en el la lista correspondiente al indice requerido.

Primero se usa el metodo *get()* al cual se agrega como parametro la variable en **index** lo que nos devuelve la lista en el indice que se encuentre en **index**, despues de eso usamos el metodo *add()* que usa de parametro el elemento en **element** para agregar ese elemento a la lista que corresponda al indice en **index**.

Es asi como se ingresan los elementos al azar en las listas.

Despues de esto hacemos uso de el metodo *print()* el cual dentro de su cuerpo, simplemente itera a traves de cada uno de los indices de **masterList** los cuales corresponde a cada una de las listas de enteros que esta contiene y despues por medio del metodo *System.out.println()* en forma estructurada, se indica cual es el numero de lista que se esta imprimiendo y los contenidos de esta. Esto sucede cada momento despues de que se agrega un elemento para ir mostrando como va progresando este codigo.

Al finalizar el programa, simplemente regresa al menu principal de la clase *Encadenamiento* y el usuario es quien decide si continua ingresando elementos o usa la opcion 2 para salir de la instancia de clase.

6.2 Ejecucion

En las siguientes imagenes, se muestra la ejecucion de la clase *Encadenamiento*.

```
1) Manejo de Tablas Hash en Java
2) Función hash por módulo
3) Encadenamiento
4) Salir
3
1) Agregar elemento
2) Salir
```

Ejercicio 3: Menu Principal


```
Ingresa elemento a agregar:
```

```
5
```

```
Lista 1 : []
```

```
Lista 2 : []
```

```
Lista 3 : []
```

```
Lista 4 : []
```

```
Lista 5 : []
```

```
Lista 6 : []
```

```
Lista 7 : []
```

```
Lista 8 : []
```

```
Lista 9 : []
```

```
Lista 10 : []
```

```
Lista 11 : []
```

```
Lista 12 : []
```

```
Lista 13 : [5]
```

```
Lista 14 : []
```

```
Lista 15 : []
```

Ejercicio 3: Opcion 1, se agrega elemento 5

```
1
Ingrese elemento a agregar:
84
Lista 1 : []
Lista 2 : []
Lista 3 : []
Lista 4 : []
Lista 5 : []
Lista 6 : []
Lista 7 : [84]
Lista 8 : []
Lista 9 : []
Lista 10 : []
Lista 11 : []
Lista 12 : []
Lista 13 : [5]
Lista 14 : []
Lista 15 : []
```

Ejercicio 3: Opcion 1, se agrega elemento 84

```
1
Ingrese elemento a agregar:
5487
Lista 1 : []
Lista 2 : []
Lista 3 : []
Lista 4 : []
Lista 5 : []
Lista 6 : []
Lista 7 : [84]
Lista 8 : []
Lista 9 : []
Lista 10 : []
Lista 11 : [5487]
Lista 12 : []
Lista 13 : [5]
Lista 14 : []
Lista 15 : []
```

Ejercicio 3: Opcion 1, se agrega elemento 5487

```
1
Ingrese elemento a agregar:
86151
Lista 1 : []
Lista 2 : [698494]
Lista 3 : []
Lista 4 : [623, 87451]
Lista 5 : [32, 4785, 86151]
Lista 6 : []
Lista 7 : [84, 23, 2189]
Lista 8 : [62, 52, 6]
Lista 9 : [948]
Lista 10 : [84]
Lista 11 : [5487]
Lista 12 : [852, 9741]
Lista 13 : [5, 84]
Lista 14 : []
Lista 15 : []
```

Ejercicio 3: Despues de agregar diferentes elementos

```
1) Agregar elemento
2) Salir
2
1) Manejo de Tablas Hash en Java
2) Función hash por módulo
3) Encadenamiento
4) Salir
4

Process finished with exit code 0
```

Ejercicio 3: Opcion 2, finalizacion del programa

7 Conclusiones

Esta practica estuvo interesante, porque me ayudo a entender como implementar una funcion *Hash Table* en Java, algo que pienso ni si quiera intentado en C anteriormente.

Me parece que en *Python* ya la habia usado y creo que es este tipo de dato al que le llaman *Dictionary*, el cual sinceramente siempre me parecio bastante practico, pero que cuando al momento de que la llave no se encontraba, entonces habia que hacer por ahi algo con codigo mas elaborado para poder evitar ese error.

Por otro lado durante la practica, escribir el codigo no se me hizo muy complicado. Fue de hecho en el ejercicio 3 donde pense que iba a ser algo mas complejo, puesto que *IntelliJ IDEA* al momento de declarar una Lista me arrojaba una lista kilométrica de metodos que no sabia ni que hacer con ellos. De hecho habia creado un par de clases adicionales para lidiar con las listas, pero despues de un poco de investigacion, encontre una forma mas rapida de realizar el ejercicio.

Algo que no sabia y que pude ver en las clases en linea, es que al momento en que se declara una variable de cualquier tipo de clase, antes del *new* lo unico que se crea es una referencia y no el objeto como tal. El objeto o la instancia ya viene despues y este se crea precisamente usando *new* para hacerlo. Pero algo particular de esto es que ese objeto se crea en la memoria y la referencia es unicamente el codigo que "apunta" hacia el objeto, precisamente como lo hacen los apuntadores en C.

Por esta razon me di cuenta, que no habia que agregar 15 listas diferentes al ejercicio 3, sino que solamente con crear distintas referencias asignadas a distintos objetos se podia lograr lo necesario para poder completar el ejercicio.

Creo que fue el ejercicio que me gusto mas de los 3.

Muchas gracias por leer mi practica!