	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: EDGAR TISTA GARCÍA

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS II

Grupo: 9

No de Práctica(s): 5

Integrante(s): CARRICHI DE LA CRUZ ROBERTO CARLOS

No. de Equipo de cómputo empleado: TRABAJO EN CASA

No. de Lista o Brigada: 07

Semestre: 2021-1

Fecha de entrega: NOVIEMBRE 06, 2020

Observaciones:

CALIFICACIÓN: _____

OBJETIVO:

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

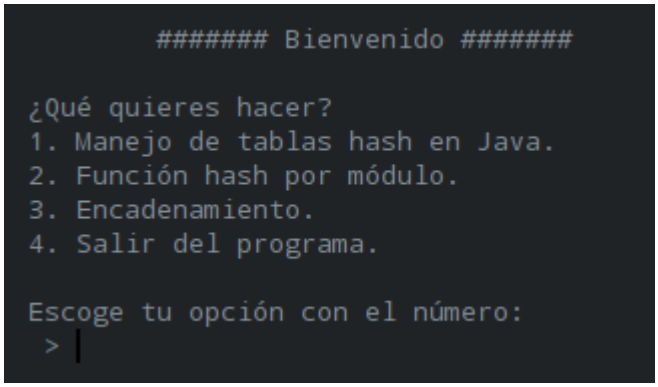
OBJETIVO DE CLASE:

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

ACTIVIDADES DEL DESARROLLO DE LA PRÁCTICA

Ejercicio 0 - Menú para la práctica
a) Con ayuda de NetBeans, crea un nuevo proyecto llamado Practica5[ApPaternoNombre].
b) Crea un menú en el cual puedas seleccionar entre las opciones siguientes

- **1)** Manejo de Tablas Hash en Java
- **2)** Función hash por módulo
- **3)** Encadenamiento



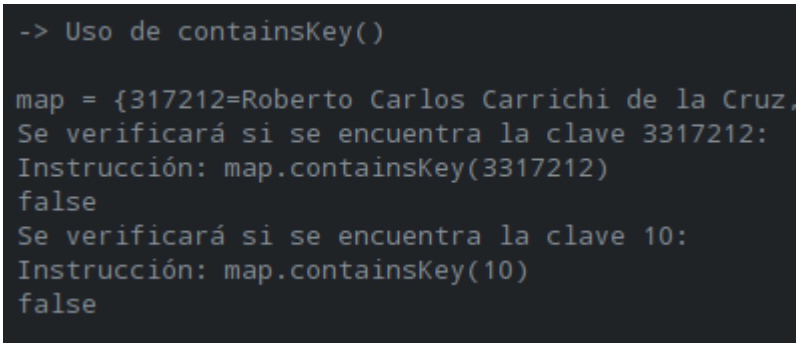
Ejercicio 1 - Tablas Hash en Java

Deberás crear una TablaHash en java (puede ser HashMap HashTable), para la tabla hash deberás crear parejas donde el valor será un “String” que representa el nombre (completo) de un alumno y la clave de la tabla hash será un numero entero que representa el número de cuenta.

Prueba los siguientes métodos y explica su funcionamientos:

- containsKey( )

Este método implementa un búsqueda en la que el objetivo a encontrar es una llave en el caso de las Tablas Hash, se refiere a el primer valor, este devolverá un valor booleano dependiendo si se encontró la llave o no.



- containsValue( )

A diferencia del método containsKey( ), este método buscará en la tabla Hash mediante los valores, devolviendo también un valor booleano dependiendo del resultado de la búsqueda, true para un caso afirmativo y false para un caso negativo.

```
-> Uso de containsValue()

map = {317212=Roberto Carlos Carrichi de la Cruz, 523213=Dulce Carrichi de la Cruz,
Se verificará si se encuentra el valor "Roberto Carlos Carrichi de la Cruz":
Instrucción: map.containsValue("Roberto Carlos Carrichi de la Cruz")
true
Se verificará si se encuentra el valor "Edgar Tista García":
Instrucción: map.containsValue("Edgar Tista García")
false
```

- equals( )

El uso de este método es para comparar dos elementos completos, aquí debe cumplirse extrictamente que todos los elementos sean idénticos, si alguno tiene más o tiene un valor cambiado se devolverá un false y un true para el caso contrario.

```
        # Uso de equals() #
Se verificará si son iguales dos mapas:
El contenido de map1 es: {123456=Juan Carlos Torres Pérez}
El contenido de map2 es: {123456=Juan Carlos Torres Pérez}
Se verificará si son iguales dos mapas:
Instrucción: map1.equals(map2)
true
Se añadirá la clave y registro <312327, "Jesus Razo Gutiérrez"> al contenido de map2 y se llamará a la función de nuevo.
Instrucción: map2.put(312327,"Jesus Razo Gutiérrez")
Instrucción: map1.equals(map2)
false
```

- get( )

Se puede observar que una parte de su funcionamiento es la de búsqueda. Como argumento del método se manda una clave y lo que se devolverá ahora será el contenido del “registro”, ya no un valor booleano. En dado caso que el valor no se encuentre enviará un valor nulo.

```
        # Uso de get() #
Se tratará de obtener el valor que tenga la clave 317212, (Se sabe que el "registro" existe.
Intrucción: map.get(317212)
Roberto Carlos Carrichi de la Cruz
Se tratará de obtener el valor que tenga la clave 10. (Se sabe que no existe ese "registro")
Intrucción: map.get(10)
null
```

- put( )

Este método es de los más utilizados y podría interpretarse como el equivalente de add( ), pero este **NO** se organizará en orden, por la misma razón no llevan el mismo nombre, pero la escencia del método es la misma: agregar elementos a la estructura. Extrictamente se requerirá de ingresar una clave y un valor para el correcto funcionamiento del método.

```
        # Uso de put() #
Se añadirá un elemento a la tabla Hash con los valores <100932, "Edgar Tista García">
Intrucción: map.put(100932, "Edgar Tista García")
Se comprobará la existencia de la clave y el valor.
Intrucción: map.containsKey(100932)
true
Intrucción: map.containsValue("Edgar Tista García")
true
```

- remove( )

La escencia de este método es sencilla, eliminar elementos del a estructura. Pero se requeriá solo de la clave para también eliminar el contenido del elemento (conocido como valor). Si se solicita imprimir el resultado de la instrucción se imprimirá un valor nulo, por lo tanto, el método no expresa ningún valor de retorno.

```
        # Uso de remove() #
Se eliminará el "registro" que tenga la clave 4.
En este ejemplo el registro es: <141517,"Claudia Cruz García">
Intrucción: map.remove(141517)
Se eliminó el valor: null
Se verificará que si el registro existe en la tabla Hash.
Intrucción: map.containsValue("Claudia Cruz García")
true
```

- size( )

El valor de retorno de este método es de tipo entero y representará la cantidad de elementos que tiene la estructura hasta el momento en que se ejecuta la instrucción, si se añaden o eliminan elementos este valor también es afectado.

```
        # Uso de size() #
Se obtendrá la cantidad de "registros" que tiene la tabla Hash
Intrucción: map.size()
7
Se añadirá un elemento y se verificará de nuevo la cantidad de "registros"
Intrucción: map.put(168728,"Daniela Torres Zurita")
Intrucción: map.size()
8
```

- clear( )

Es de los pocos métodos que no requiere de atributos para su funcionamiento, esta función se dedicará a eliminar cada uno de los “registros ” que tiene nuestra estructura. Este método no proporciona ningún valor de retorno, por lo tanto, la información no puede ser recuperada.

```
        # Uso de clear() #
El contenido actual de la tabla es: {123456=Juan Carlos Torres Pérez, 312327=Jesus Razo Gutiérrez}
Se limpiará el contenido de la tabla.
Instrucción: map2.clear()
El contenido final de la tabla es: {}
```

- isEmpty( )

Por último, este método buscará identificar si la estructura se encuenta vacía. No requerirá de ningún parámetro y siembre retornará un resultado de tipo booleano.

```
        # Uso de isEmpty() #
map = {16=Daniela Torres Zurita, 100932=Edgar Tista García, 317212=Roberto Carlos Carrichi de la Cruz,
Se verificará si se encuentra vacía.
Intrucción: map.isEmpty()
false
Se eliminará el contenido y se volverá a verificar.
Intrucción: map.clear()
Intrucción: map.isEmpty()
true
```

Ejercicio 2 - Simulación de Función Hash por módulo.

a) Agrega una clase al proyecto llamada HashMódulo, en ella, crea una nueva lista (puede ser linked list o array list) de enteros; el tamaño de la lista se definirá desde un principio. Para efectos de la práctica será de 15 (para crear una lista de 15 elementos debes inicializar todos sus elementos con null)

Elabora un submenú para que el usuario pueda seleccionar entre las siguientes opciones**:**

a) Agregar elementos
b) Imprimir lista
c) Buscar elementos

```
         #### Función hash por módulo ####

        ¡Se acaba de crear una lista de 15 elementos para ti!

¿Qué deseas hacer?
        1. Agregar elementos.
        2. Imprimir lista.
        3. Buscar elementos.
        4. Volver al menú principal.

Escoge tu opción con el número:
 >
```

b) Crea un método donde se utilizará la función hash por módulo cada vez que el usuario agregue o busque elementos en la lista. Considera lo siguiente:

- ✔ El valor de M para la función hash por módulo será un valor M=13.

Para el desarrollo de este punto se creó un método dedicado a encontrar los índices dando como segundo argumento la cantidad de módulo, para que esa más fácil de modificar si se decide realizar una posible mejora pidiendo más valores al usuario.

```
int indice = encontrarIndice(valor,13);
```

- ✔ Cada vez que el usuario agregue un nuevo valor, la función hash calculará el índice que ocupará en la lista ese valor.

```
Ingresa el valor que quieres agregar:
 > 14
El valor se guardará en el índice: 1.
¡Agregado con éxito! (1/2)
```

- ✔ En caso de haber colisiones, ser resolverán por el método de PRUEBA LINEAL.

```
Ingresa el valor que quieres agregar:
 > 14
¡COLISIÓN! Ya está ocupada la posición con el índice 1.
El valor se guardará en el índice: 2.
¡Agregado con éxito! (2/2)
```

c) Al agregar o buscar elementos el programa deberá informar al usuario el resultado de la función hash aplicada indicando la posición donde se almacenó/encontró el elemento.

```
        #### Buscar elementos ####

Escribe la cantidad que estás buscando:
 > 14
¡El número ha sido encontrado!
La cantidad que ingresaste se encuentra en el índice 1.
```

Ejercicio 3. Encadenamiento

1. Agrega una nueva clase al proyecto llamada Encadenamiento, en ella deberás realizar lo siguiente:

a) Deberás crear una lista de 15 listas de enteros.

Para lograr implementar esta lista tuvo que declararse la lista principal de la siguiente forma:

```
ArrayList<ArrayList<Integer>> lista = new ArrayList<ArrayList<Integer>>();
```

Hasta el momento, la lista ya tenía definido que tipo de objetos iba a contener, pero aún no tenía la extensión de 15 listas, por lo que se debía de extender al menos una vez con un valor nulo. En el desarrollo de este ejercicio, se hizo uso de un método dedicado a “rellenar” las 15 posiciones con un valor nulo. Se decidió hacerce así con el fin de hacer más sencilla una posible mejora en la que el usuario agrege datos desde el teclado.

```
extenderLista(lista,15);
```

b) Implementa una función aleatoria que determinará la posición de los elementos ingresados por el usuario.

Para fines prácticos también se utilizó un método dedicado a buscar un número aleatorio entre 0 y 14 (los posibles índices de nuestra lista principal), también haciendo que se lograra tener un código mejor estructurado. Dado que se tratada de un resultado aleatorio, no requiere de argumentos para que funcione el programa correctamente.

```
int indice = encontrarIndice();
```

2. El usuario podrá elegir, agregar elemento o salir de la sección. Cada vez que el usuario ingrese un número, la función aleatoria indicará la posición donde se ubicará el número y al final mostrará el estado global de la lista.

En la ejecución del programa, puede notarse que la “Lista derivada” hace referencia cada una de las listas que contiene la lista principal, expresando mejor la estructura como renglones y los valores que va añadiendo el usuario se colocan como columnas en la lista que le corresponde según el resultado de la función aleatoria.

```
Ingresa el valor que quieres agregar:
 > 983
Se agregará en la lista con el índice: 4
¡Valor añadido con éxito!  (1/1)

Has terminado de añadir.
El estado de la lista es el siguiente:
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [983]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
Lista derivada -> [ ]
```

Elabora las conclusiones de tu práctica indicando si se lograron los objetivos de aprendizaje.

Al realizar práctica se logró entender a profundidad el funcionamiento de la tranformación de llaves para poder ordenar elementos y así también lograr que los métodos de búsqueda fueran los más eficientes posibles. No se presentó tanta complicación al realizar la implementación de las transformaciones de llaves, en realidad, la dificultad depende de la forma en la que se abstrae el problema y como se puede aplicar modularidad para hacerlo más entendible. El desarrollo de estos ejercicios también ayudó a crear bases más firmes para entender el funcionamiento de las listas que son capaces de contener otras listas, asímismo, se comprendió en su totalidad la forma en que se aplican distintos métodos en las tablas Hash, dado como resultado de alguna manera una extensión de conocimiento del alumno para crear programas más complejos y más eficientes.