	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	TISTA GARCÍA EDGAR
Asignatura:	ESTRUCTURA DE DATOS Y ALGORITMOS II
Grupo:	5
No de Práctica(s):	PRACTICA #4 – ALGORITMOS DE BUSQUEDA
Integrante(s):	CARRILLO CERVANTES IVETTE ALEJANDRA
No. de Equipo de cómputo empleado:	TRABAJO EN CASA
No. de Lista o Brigada:	09
Semestre:	2022 - 1
Fecha de entrega:	13 OCTUBRE 2021
Observaciones:	

CALIFICACIÓN: _____

PRÁCTICA #4 – ALGORITMOS DE BÚSQUEDA PARTE 1

Objetivo: El estudiante identificará el comportamiento y características de los principales algoritmos de búsqueda por comparación de llaves.

Objetivo de clase: El alumno aplicará la búsqueda por comparación de llaves mediante la implementación de listas de tipos de datos primitivos y de tipos de datos abstractos.

ACTIVIDADES PARA EL DESARROLLO DE LA PRÁCTICA

Ejercicio 1. Listas en Java.

Este ejercicio tiene como objetivo presentar los métodos más importantes de la biblioteca "LinkedList", así como probar cada uno con una lista con elementos tipo Integer, los cuales se irán modificando con forme vaya avanzando el programa.
Se realizó una pequeña investigación acerca de LinkedList, así como de sus métodos:

LinkedList construye una lista que contiene los elementos de la colección especificada, en el orden en que los devuelve el iterador de la colección.

En esta práctica se utilizaron los siguientes métodos:

- *add()*

Es un método sobrecargado, tiene como objetivo agregar un valor a la lista creada.
  - add(elemento). Agrega el elemento especificado al final de la lista.
  - add(posición, elemento). Agrega el elemento en una posición determinada de la lista.

- *set(indice, elemento)*

Este método reemplaza el elemento en la posición especificada, por el elemento especificado.

- *subList(indice1, indice2)*

Es un método que devuelve una vista de la parte de una lista entre dos posiciones especificadas.

- *remove()*

Es un método sobrecargado, tiene como objetivo eliminar un elemento de la lista.
  - remove(posición). Elimina el elemento en la posición especificada.
  - removeFist(). Elimina y devuelve el primer elemento de la lista.
  - removeLast(). Elimina y devuelve el último elemento de la lista.

- *isEmpty()*.

Devuelve verdadero si esta lista no contiene elementos.

- *indexOf(Objeto o)*

Devuelve el índice de la primera aparición del elemento especificado en la lista.

Como se mencionó anteriormente, para probar estos métodos se realizó un “LinkedList” con elementos de tipo Integer, con el objetivo de ir probando cada método investigado; aparte, con ayuda de otro método se imprimió la lista cada vez que se hacía alguna modificación. La salida de este programa es el siguiente:

```
run:
******
 Estado 1
Se agregaron 7 elementos a la lista:
15
80
16
6
24
31
26
******
 Estado 2
Se agregaron 3 elementos en posiciones especificadas de la lista:
15
80
300
16
500
700
6
24
31
26
******
 Estado 3
Se reemplazaron elementos en posiciones especificadas de la lista:
15
14
16
16
500
700
6
18
31
26
******
 Estado 4
Se eliminó el elemento en la posición 1 de la lista...
Se eliminó el elemento en la posición 6 de la lista...
15
16
16
500
700
6
31
26
******
 ¿La lista 1 esta vacia? false
******
 Se busca el elemento 6 en la lista...
 El elemento 6, esta en la posición: 5
 Se busca el elemento 16 en la lista...
 El elemento 16, esta en la posición: 1
******
Se creó la lista 2 y 3 con respecto a la lista 1
 Lista 2
16
500
******
 Lista 3
16
500
******
¿La lista 1 es igual a la lista 2?
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Practica4_[Carrillo][Ivette] -> ./Listas.java

Busqueda lineal

Para implementar este algoritmo, se creo una clase diferente de la que contiene el método main, en ella se crearon los siguientes métodos:

→ encontrar()
Tiene como parámetros la lista en donde se buscará el elemento y dicho elemento, dentro de este método se encuentra un ciclo de repetición “for-each” el cual recorrerá toda la lista e irá comparando si el valor de la posición donde se encuentra es igual al numero a buscar, en dado caso de que se cumpla la condición, retornará un true; sin embargo, en dado caso de que haya recorrido toda la lista y nunca se encuentre el elemento, retornara un false.

→ indice()
Tiene como parámetros la lista en donde se buscará el elemento y dicho elemento, este método se hizo muy similar al anterior, pues se fue recorriendo la lista con un ciclo de repetición, ahora “for”, y se hizo la misma comparación; sin embargo, en esta ocasión en vez de que se retorne un valor booleano para saber si se encuentra o no el elemento, se retornará el indice donde se cumplió la igualdad, en dado caso de que no se cumpla la igualdad, retornará como indice el número “-1”.

→ numVeces
Tiene los mismos parámetros que los métodos anteriores, se recorrió la lista y se hizo la misma comparación; no obstante, esta vez se inicializo en cero una variable tipo int al principio del método, con el objetivo de que al momento de recorrer la lista, cada que se cumpliera la condición de que si el elemento en la posición donde se encuentra el for, es igual al número por buscar, iría aumentando 1 la variable la cual se inicializó en cero. Finalmente, se retorna dicha variable, ya que contiene el número de veces que se encontró un elemento.

→ Lineal
Tiene como parámetro la lista en la que se buscará el elemento, dentro de este método se le pregunta al usuario cual es el valor que se desea buscar, después de hace la comparación de que si el valor de retorno del método encontrar, es verdadero, entonces se llamará a los métodos indice y numVeces, posteriormente se imprimirá los datos que retornan estos métodos (posición donde se encuentra el elemento y el número de veces que aparece, respectivamente); sin embargo, si lo que devuelve el método encontrar es falso, se imprimirá al usuario que el valor no se encuentra en la lista.

Busqueda Binaria

Para implementar este algoritmo, al igual que en Busqueda Lineal, se creó una clase diferente de la que contiene el método main, en ella se crearon los siguientes métodos:

→ encontrarIndice
Tiene como parámetro la lista en donde se buscará el elemento y el elemento a buscar, para implementar este pedazo de código se siguió el pseudocódigo visto en clase, se quiso implementar el pseudocódigo recursivo; sin embargo, marcaba error al hacer la llamada recursiva, por lo tanto, se utilizó el pseudocódigo normal. Este método compara el valor buscado con el que ocupa la posición central y si no son iguales se divide la lista a la mitad según si es mayor el elemento central al buscado o no, esto se va repitiendo hasta tener una lista de tamaño 1; finalmente, retorna el valor de la posición donde se encuentra el valor buscado o un “-1” en dado caso de que no se encuentre dicho elemento.

→ numVeces
Tiene como parámetro la lista donde se buscó el elemento, el elemento que se buscó y el indice donde se encontró. En este método se crean dos variables “derecha” e “izquierda” inicializadas en el valor del indice donde se encontró el elemento; cada variable se fue recorriendo a la izquierda o a la derecha según sea el caso y verificando si el elemento siguiente o anterior es igual al valor a buscar, y en dado caso de que sea verdadero se fue aumentando a 1 el valor de una variable contador previamente inicializada en cero, finalmente, se retorna el valor del contador.

→ Binaria
Tiene como parámetro la lista donde se quiere buscar el valor, en este método se le pregunta al usuario que valor desea buscar en la lista, se manda a llamar al metodo encontrar indice y entra a una estructura condición la cual indica que si el retorno del método encontrarIndice es “-1”, es decir no se encontró, se imprimirá que el valor no se encuentra, pero en caso contrario que si se encuentre el valor, mandará a llamar al método numVeces y se imprimirá la posición y el número de veces que se encuentra el elemento a buscar.

Se creó un nuevo archivo llamado “*BusquedaLinealyBinaria*” con el objetivo de probar los métodos realizados para la Busqueda Lineal y Binaria, en este se crearon dos listas para ocupar una en cada busqueda. Para Busqueda Lineal se creó un método donde la lista de tamaño que ingresará el usuario se llenaría con números aleatorios y con base a esta se llamaría al método “*Lineal*” que se creó en la clase “*Busqueda Lineal*”; mientras que, para la Busqueda Binaria, también se creó un nuevo método el cual también llenaría la lista de tamaño que ingresará el usuario; sin embargo, para la busqueda binaria debemos de tener la lista ordenada, así que en este mismo método se mando a llamar a el método “*QuickSort*” de la clase “*QuickSort*” (creada basándonos en prácticas anteriores de algoritmos de ordenamiento) con el fin de que la lista quedará ordenada, y una vez ordenada se llamaría al método “*Binaria*” que se creó en la clase “*BusquedaBinaria*”.

Para comprobar su funcionamiento, se hicieron dos ejecuciones, una buscando un elemento que se encuentre en la lista y otra buscando un elemento que no exista en la lista.

```
run:
__  __  __  __  __  BUSQUEDA LINEAL  __  __  __  __  __
¿De cuántos elementos quieres generar tu lista aleatoria? 25
52 - 75 - 16 - 3 - 44 - 39 - 51 - 91 - 88 - 90 - 6 - 85 - 97 - 25 - 81 - 67 - 21 - 86 - 4 - 42 - 38 - 47 - 83 - 42 - 66
¿Qué valor deseas buscar en la lista? 97
        * * * * * * * * * * * * * * * * * * * * * * * * * *
        * SI se encuentra el valor 97 en la lista (:    *
        * Posición: 12                                   *
        * Número de veces: 1                            *
        * * * * * * * * * * * * * * * * * * * * * * * * * *

__  __  __  __  __  BUSQUEDA BINARIA  __  __  __  __  __
¿De cuántos elementos quieres generar tu lista aleatoria? 25
1 - 1 - 2 - 2 - 2 - 2 - 3 - 3 - 4 - 5 - 5 - 5 - 6 - 6 - 6 - 7 - 7 - 8 - 8 - 8 - 8 - 9 - 9 - 9 - 10 -
¿Qué valor deseas buscar en la lista? 6
        * * * * * * * * * * * * * * * * * * * * * * * * * *
        * SI se encuentra el valor 6 en la lista (:    *
        * Posición: 12                                   *
        * Número de veces: 3                            *
        * * * * * * * * * * * * * * * * * * * * * * * * * *
BUILD SUCCESSFUL (total time: 51 seconds)
```

Primera ejecución Practica4_[Carrillo][Ivette] -> ./BusquedaLinealyBinaria.java

```
run:
__  __  __  __  __  BUSQUEDA LINEAL  __  __  __  __  __
¿De cuántos elementos quieres generar tu lista aleatoria? 25
24 - 34 - 77 - 25 - 52 - 51 - 13 - 42 - 80 - 10 - 1 - 25 - 40 - 79 - 41 - 59 - 26 - 1 - 11 - 27 - 63 - 68 - 55 - 56 - 19
¿Qué valor deseas buscar en la lista? 97
        * * * * * * * * * * * * * * * * * * * * * * * * *
        *   El valor no se encuentra en la lista ):     *
        * * * * * * * * * * * * * * * * * * * * * * * * *

__  __  __  __  __  BUSQUEDA BINARIA  __  __  __  __  __
¿De cuántos elementos quieres generar tu lista aleatoria? 25
1 - 1 - 2 - 2 - 2 - 3 - 4 - 4 - 4 - 4 - 4 - 5 - 5 - 6 - 6 - 6 - 6 - 6 - 6 - 7 - 8 - 9 - 9 - 10 - 10 -
¿Qué valor deseas buscar en la lista? 12
        * * * * * * * * * * * * * * * * * * * * * * * * *
        *   El valor no se encuentra en la lista ):     *
        * * * * * * * * * * * * * * * * * * * * * * * * *
BUILD SUCCESSFUL (total time: 25 seconds)
```

Segunda ejecución Practica4_[Carrillo][Ivette] -> ./BusquedaLinealyBinaria.java

Ejercicio 4. Búsqueda en listas de objetos.

Este ejercicio tiene como finalidad probar tanto la busqueda lineal como la busqueda binaria en una lista de objetos, en este caso Computadoras. Para ello se creó una clase llamada "Computadora", la cual tendrá 3 métodos constructores: un método vacío, un método que recibe como parámetros la marca y la memoria ram de la computadora y otro que recibe todos los atributos de la computadora; también esta clase contará con los métodos de acceso de cada atributo y con los siguientes dos métodos de clase:
→ descuento. Tiene como función obtener el 20% de descuento de una computadora si se cumple la condición de que su atributo "descuento" es verdadero, en dado de que sea falso se imprimirá un aviso que la computadora no tiene ningún descuento.
→ imprimirComputadora. Tiene como función imprimir los atributos de una lista de computadoras la cual se pasa como parámetro, al momento de imprimir si la computadora tiene descuento o no, se manda a llamar al método descuento.

En otro archivo llamado "*PrincipalComputadora*" se inicializaron 5 computadoras con sus respectivos atributos y se agregaron a una lista de Computadoras creada previamente:

```
run:

  * Se inicializaron las siguientes 5 computadoras antes de iniciar el programa *
__  __  __  __  __  __  __  __  __  __  __

Marca: Apple
Memoria RAM: 8 GB
Color: blanco
Precio: 32000.0
La computadora no tiene ningún descuento ):
__  __  __  __  __  __  __  __  __  __  __

Marca: Dell
Memoria RAM: 4 GB
Color: negro
Precio: 19000.0
La computadora tiene un descuento del 20%
Su precio final es: 15200.0
```

```
— — — — — — — — — — —

Marca: Huawei
Memoria RAM: 8 GB
Color: gris
Precio: 22000.0
La computadora no tiene ningún descuento ):
— — — — — — — — — — —

Marca: HP
Memoria RAM: 16 GB
Color: gris
Precio: 28000.0
La computadora tiene un descuento del 20%
Su precio final es: 22400.0
— — — — — — — — — — —

Marca: Lenovo
Memoria RAM: 8 GB
Color: gris
Precio: 24000.0
La computadora tiene un descuento del 20%
Su precio final es: 19200.0
— — — — — — — — — — —
```

Impresión de las 5 computadoras creadas en un principio

Posterior a ello, se creó un menú de opciones el cual se repetirá hasta que el usuario decida salir, en dicho menú se tiene lo siguiente:

➔ Crear una Computadora

Esta opción tiene un pequeño menú el cual le pregunta al usuario si quiere crear una computadora con todos sus atributos (para lo cual se utiliza el tercer método constructor de la clase computadora) o bien, quiere crear una computadora solo inicializando la marca y la memoria RAM de esta (para lo cual se utiliza el segundo método constructor de la clase computadora). Sin importar la opción que elija, se creará una nueva computadora y se agregará a la lista de computadoras con el método “add()”.

```
_____________________________________________

                MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 1
______  ______  ______  ______  ______  ______

        CREAR UNA COMPUTADORA

 1 -> Crear con todos sus atributos
 2 -> Crear solo con marca y memoria RAM (los demás atributos inicializarlos en cero)
Elige una opción: 1

Inserta los datos de la computadora:
Marca: Apple
Memoria RAM: 8
Color: gris
Precio: 32500
Descuento (true/false): true
```

Primera opción del menú al elegir crear computadora con todos sus atributos

```
_____________________________________________

               MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 1
______  ______  ______  ______  ______  ______

        CREAR UNA COMPUTADORA

 1 -> Crear con todos sus atributos
 2 -> Crear solo con marca y memoria RAM (los demás atributos inicializarlos en cero)
Elige una opción: 2

Inserta los datos de la computadora:
Marca: HP
Memoria RAM: 8
```

Primera opción del menú al elegir crear computadora solo con dos atributos

➔ Imprimir lista de computadoras

Esta opción tiene como función imprimir la lista donde se van almacenando las computadoras, para ello se manda a llamar al método imprimirComputadoras de la clase Computadora, pasándole como parámetro la lista.

```
_____________________________________________

               MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 2
______  ______  ______  ______  ______  ______

        IMPRIMIR COMPUTADORAS
__  __  __  __  __  __  __  __  __  __  __

Marca: Apple
Memoria RAM: 8 GB
Color: blanco
Precio: 32000.0
La computadora no tiene ningún descuento ):
__  __  __  __  __  __  __  __  __  __  __

Marca: Dell
Memoria RAM: 4 GB
Color: negro
Precio: 19000.0
La computadora tiene un descuento del 20%
Su precio final es: 15200.0
__  __  __  __  __  __  __  __  __  __  __
```

```
— — — — — — — — — — —

Marca: Huawei
Memoria RAM: 8 GB
Color: gris
Precio: 22000.0
La computadora no tiene ningún descuento ):
— — — — — — — — — — —

Marca: HP
Memoria RAM: 16 GB
Color: gris
Precio: 28000.0
La computadora tiene un descuento del 20%
Su precio final es: 22400.0
— — — — — — — — — — —

Marca: Lenovo
Memoria RAM: 8 GB
Color: gris
Precio: 24000.0
La computadora tiene un descuento del 20%
Su precio final es: 19200.0
— — — — — — — — — — —

Marca: Apple
Memoria RAM: 8 GB
Color: gris
Precio: 32500.0
La computadora tiene un descuento del 20%
Su precio final es: 26000.0
— — — — — — — — — — —

Marca: HP
Memoria RAM: 8 GB
Color: null
Precio: 0.0
La computadora no tiene ningún descuento ):
— — — — — — — — — — —
```

Segunda opción del menú

➔ Buscar computadoras por marca

Para esta opción se realizó una busqueda lineal, se creó un método nuevo en la clase Computadora, el cual se llama "*LinealComputadora*" el cual recibirá la lista en donde se buscará si se encuentra la marca dada por el usuario y dicha marca, para este método se utilizó un ciclo "*for each*" el cual irá recorriendo la lista y comparando el atributo de el objeto en la posición donde se encuentra con la marca que ingreso el usuario, si esta condición se cumple, entonces se agregará el objeto Computadora a una lista nueva la cual contendrá las computadoras donde coinciden sus marcas y posteriormente se retornará.

```
_____________________________________________

                MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 3
______  ______  ______  ______  ______  ______

        BUSCAR COMPUTADORAS POR MARCA

Ingresa la marca a buscar: Apple

Las computadoras que tienen como marca 'Apple' son las siguientes:
```

```
— — — — — — — — — — —

Marca: Apple
Memoria RAM: 8 GB
Color: blanco
Precio: 32000.0
La computadora no tiene ningún descuento ):
— — — — — — — — — — —

Marca: Apple
Memoria RAM: 8 GB
Color: gris
Precio: 32500.0
La computadora tiene un descuento del 20%
Su precio final es: 26000.0
— — — — — — — — — — —
```

Tercera opción del menú

➔ Buscar computadoras por marca

Para esta opción se realizó una busqueda lineal, se creó un método nuevo en la clase Computadora, el cual se llamará igual que el anterior "*LinealComputadora*" el cual recibirá la lista en donde se buscará si se encuentra la memoria RAM dada por el usuario, para este método se utilizó también un ciclo "*for each*" el cual irá recorriendo la lista y comparando el atributo del objeto en la posición donde se encuentra con la memoria RAM que ingreso el usuario, si esta condición se cumple, entonces se agregará el objeto Computadora a una lista nueva la cual contendrá las computadoras donde coinciden sus memoria RAM y posteriormente se retornará esta.

```
_____________________________________________

                MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 4
______  ______  ______  ______  ______  ______

       BUSCAR COMPUTADORAS POR MEMORIA RAM

Ingresa la memoria RAM a buscar: 8

Las computadoras que tienen una memoria RAM de 8 GB son las siguientes:
— — — — — — — — — — —

Marca: Apple
Memoria RAM: 8 GB
Color: blanco
Precio: 32000.0
La computadora no tiene ningún descuento ):
— — — — — — — — — — —

Marca: Huawei
Memoria RAM: 8 GB
Color: gris
Precio: 22000.0
La computadora no tiene ningún descuento ):
— — — — — — — — — — —

Marca: Lenovo
Memoria RAM: 8 GB
Color: gris
Precio: 24000.0
La computadora tiene un descuento del 20%
Su precio final es: 19200.0
— — — — — — — — — — —
```

```
Marca: Apple
Memoria RAM: 8 GB
Color: gris
Precio: 32500.0
La computadora tiene un descuento del 20%
Su precio final es: 26000.0
__  __  __  __  __  __  __  __  __  __  __

Marca: HP
Memoria RAM: 8 GB
Color: null
Precio: 0.0
La computadora no tiene ningún descuento ):
__  __  __  __  __  __  __  __  __  __  __
```

Cuarta opción del menú

➔ Buscar existencia de una computadora por marca

Para esta última opción se agrego el método "*BinariaComputadora*" el cual recibirá como parametros la lista donde se buscará el elemento o bien, la computadora, y el nombre de la marca que se buscará, para este ejercicio se tomó como referencia el algoritmo creado en el ejercicio 3 de esta práctica, pero se modificaron algunas cosas, como fueron las condiciones que se tienen para saber si la lista se dividirá en dos a la izquierda o a la derecha, para esta comparación, dado que son ahora tipos de datos String se utilizó el método compareTo y sus condiciones de este método. Se fue comparando el atributo marca del indice de en medio de la lista con el indice a buscar y es cero significa que son iguales, si es menor a cero significa que el String 1 es menor que el String 2 y si es mayor a cero significa que el String 1 es mayor al String2.

```
_____________________________________________

                MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 5
______  ______  ______  ______  ______  ______

        BUSCAR EXISTENCIA DE UNA COMPUTADORA

  * Solo se tomarán en cuenta las computadoras inicializadas antes del inicio del programa *

Ingresa la marca a buscar: Dell
        * * * * * * * * * * * * * * * * * * * *
        * Si se encuentra la computadora (: *
        * * * * * * * * * * * * * * * * * * * *
_____________________________________________

                MENU DE OPCIONES

  1. Crear una computadora
  2. Imprimir lista de computadoras
  3. Buscar computadoras por marca
  4. Buscar computadoras por memoria RAM
  5. Buscar existencia de una computadora por marca
  6. Salir
Elige una opción: 5
______  ______  ______  ______  ______  ______

        BUSCAR EXISTENCIA DE UNA COMPUTADORA

  * Solo se tomarán en cuenta las computadoras inicializadas antes del inicio del programa *

Ingresa la marca a buscar: Acer
        * * * * * * * * * * * * * * * * * * * *
        * No se encuentra la computadora ): *
        * * * * * * * * * * * * * * * * * * * *
_____________________________________________
```

Conclusiones

Considero que, si se cumplieron el objetivo de esta práctica, ya que realizamos ejercicios donde implementamos tanto la busqueda lineal como la busqueda binaria en tipo de datos primitivos, así como en tipos de datos abstractos. Se pudo observar que la busqueda lineal es más tardada que la busqueda binaria, puesto que debe de recorrer toda la lista e ir comparando con cada valor para saber si se encuentra el valor a buscar; mientras que en la busqueda binaria, a pesar de que se tiene la restricción de que la lista tiene que estar ordenada, considero que es más eficiente por que se basa en la estrategia de divide y vencerás.

Se cumplieron con todas las actividades de esta práctica; sin embargo, me costó un poco de trabajo implementar el algoritmo de Busqueda Binaria, pues lo quería implementar de manera recursiva y no pude (lo dejé comentado en la clase Busqueda Binaria) por lo que lo realice de forma normal, a pesar de ello, considero que las actividades de esta práctica fueron muy buenas para entender mejor los algoritmos de busqueda y también para practicar los conocimientos que se tiene en la Programación Orientada a Objetos. Personalmente, los conocimientos y experiencias adquiridos en el proyecto de Programación Orientada a Objetos me ayudaron bastante para esta práctica. (:

Referencias:

- ➔ Oracle(1993, 2020) Java SE Documentation Oracle