



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 II

Grupo: 9

No de Práctica(s): 4

Integrante(s): Díaz Hernández Marcos Bryan

*No. de Equipo de
cómputo empleado:* Equipo personal

No. de Lista o Brigada: 9

Semestre: 2021-1

Fecha de entrega: 28 de octubre de 2020

Observaciones:

CALIFICACIÓN: _____

Objetivo de la practica

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

Introducción

En el siguiente reporte se analizaran los ejercicios propuestos en la practica 4, el análisis consistirá en varias partes tanto como antes y después de realizar el ejercicio, un poco más enfocado a la parte de análisis del desarrollo del código en cada uno de los programas.

Ejercicios de la practica:

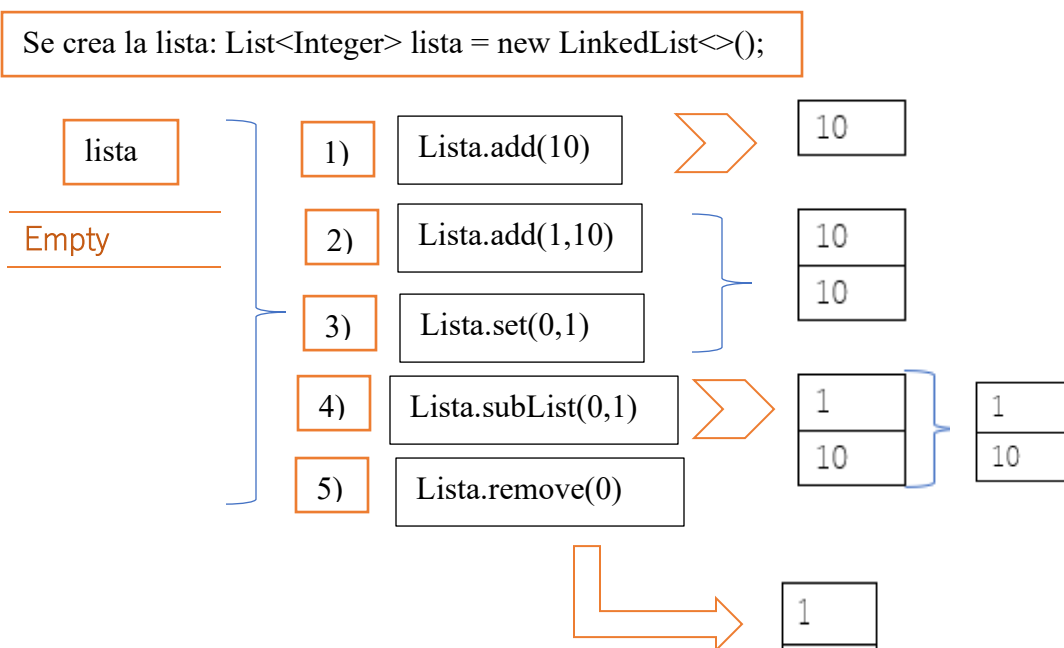
- Ejercicio 1:

El primer ejercicio consistió en implementar listas ligadas en Java, y conocer los principales métodos que estas utilizan para su manejo, fue la identificación de la clase y su implementación.

- Dificultades en el código

En este caso no tuve dificultades porque ya había visto los métodos y ya tenía idea de cómo funcionaban sin embargo algunos los desconocía y tuve que ir a la documentación que ofrece Oracle, y ahí encontré todo lo que estaba aplicando en este ejercicio.

- Diagrama de funcionamiento:



En este caso cada metodo se realiza sobre la lista, la que se modifica con caada operación, por lo que al añadir esta crece, al eliminar disminuye y al crear sublistas se envian un rango de valores a la siguiente lista.

- Relación con la teoría:

1) Diferencias entre set y add.

- Add: introduce un elemento en la posición que se indique, este debe de ser del mismo tipo de la lista
- Set: sustituye un valor en la posición que se indica, igual debe de ser del mismo tipo de la lista.

2) Explica cómo funciona el método subList.

La subList genera una lista que contiene algunos elementos de la lista principal, los elementos que contendrá serán los limitados en los index:

- `List<Integer> subList(int fromIndex, int toIndex)`

3) Investigar métodos de eliminación, verificar contenido, buscar algún elemento.

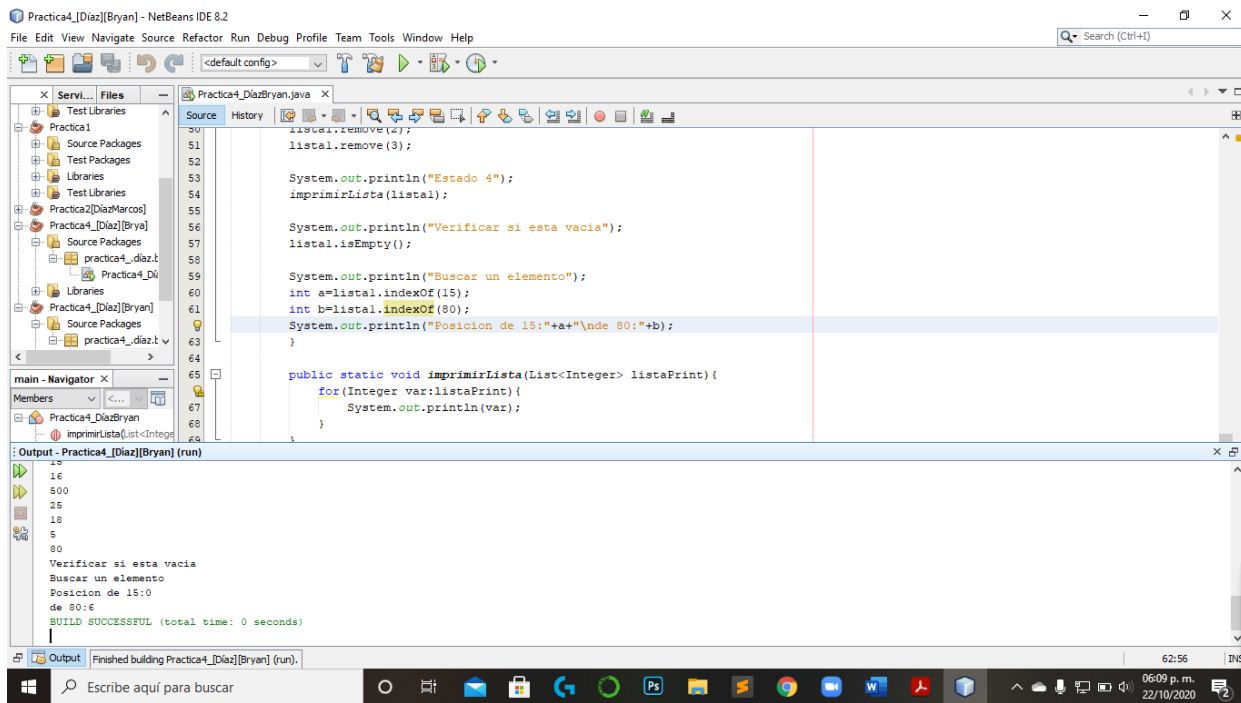
- Borrar: `remove(int index)`: remueve el elemento de la posición indicada.
- Verificar: `isEmpty()`: verifica si existen elementos en la lista, regresa un booleano
- Buscar: `indexOf(Object o)`: busca el elemento que se pone en el argumento, y regresa la posición en que se encuentra.

Al ser colecciones se tiene que son colecciones de objetos, y por ello reciben objetos como parámetro ya que están diseñadas para eso.

En cuanto a la teoría, se relaciona con la primera parte de ordenamientos, más enfocado a la búsqueda, ya que estas listas enlazadas son las que permiten un procesamiento de operaciones mucho más rápido que los ArrayList, además que son una de las tantas formas de almacenamiento dentro de las estructuras de datos.

Y porque es necesario el saber implementarlas en Java, porque en los siguientes ejercicios estas serán ocupadas para el desarrollo de cada uno, y ser primordial el reconocer como funcionan.

- Evidencia de implementación



- Ejercicio 2:

El segundo ejercicio consistía en implementar la búsqueda lineal en una lista de n elementos, que en este caso n es igual a diez, y esta búsqueda tiene tres variantes las cuales consisten en la devolución de un booleano en caso de que el valor que se busca se encuentre en la lista, la devolución de la posición del valor y por último la cantidad de veces que se encuentra el valor en la lista.

- Dificultades en el código

Lo más difícil que encontré en este ejercicio fue el crear una lista y pedir al usuario que ingresara los valores, porque me quede algo pasmado por un momento, y no sabía lo que estaba haciendo, así que no hice nada después de dos días, y cuando decidí hacerlo me di cuenta de que no está respetando las clases y más que nada las variables estáticas porque declaraba un objeto de Scanner pero dentro del método principal, e hice algo que no tenía sentido, así que decidí empezar de cero esa parte y al final solo era cuestión de definir un método estático que guardara valores en la lista por medio de un for y del método Add (Imagen 1).

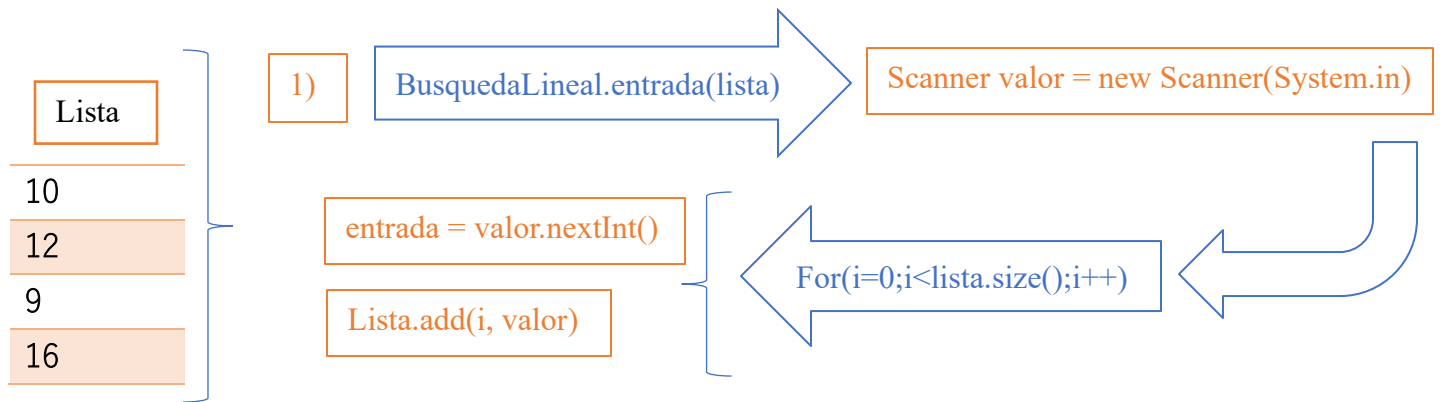
```

Scanner entrada1 = new Scanner(System.in);
for(int i=0;i<10;i++){
    System.out.println("Inserta el valor["+(i+1)+"]");
    int valor = entrada1.nextInt();
}

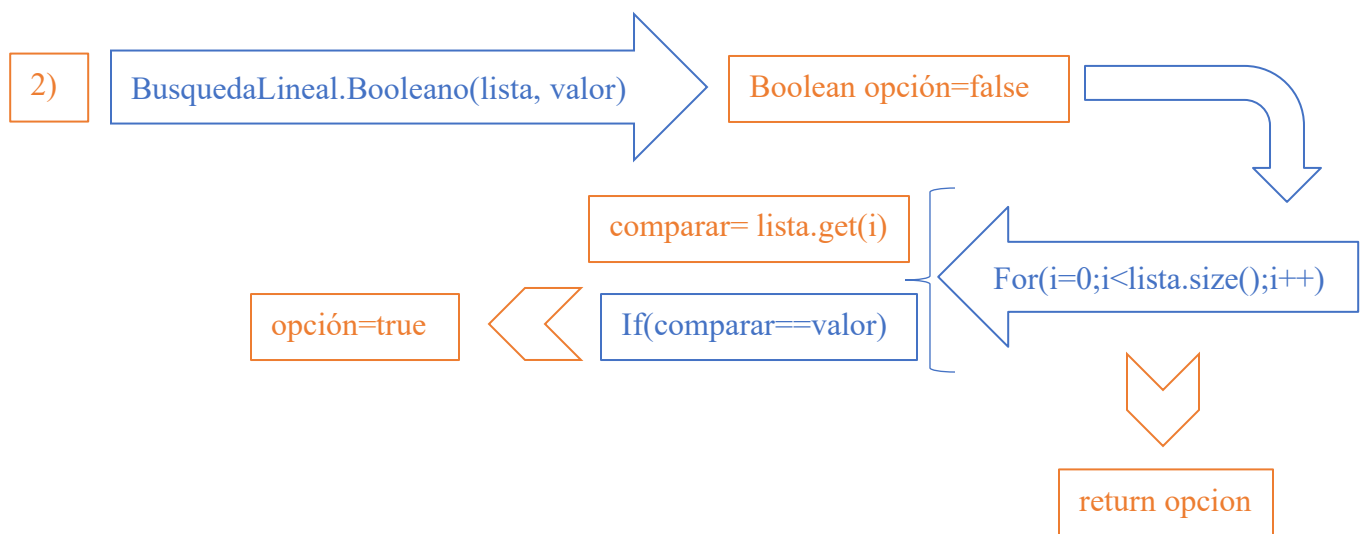
```

Imagen 1.

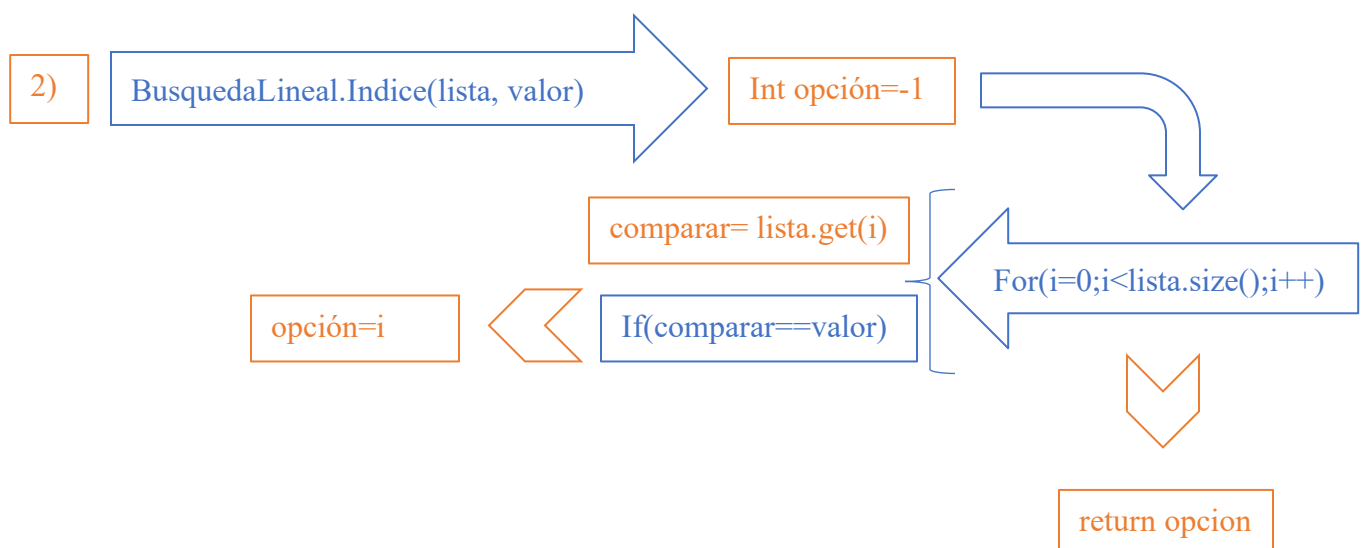
- Diagrama de funcionamiento



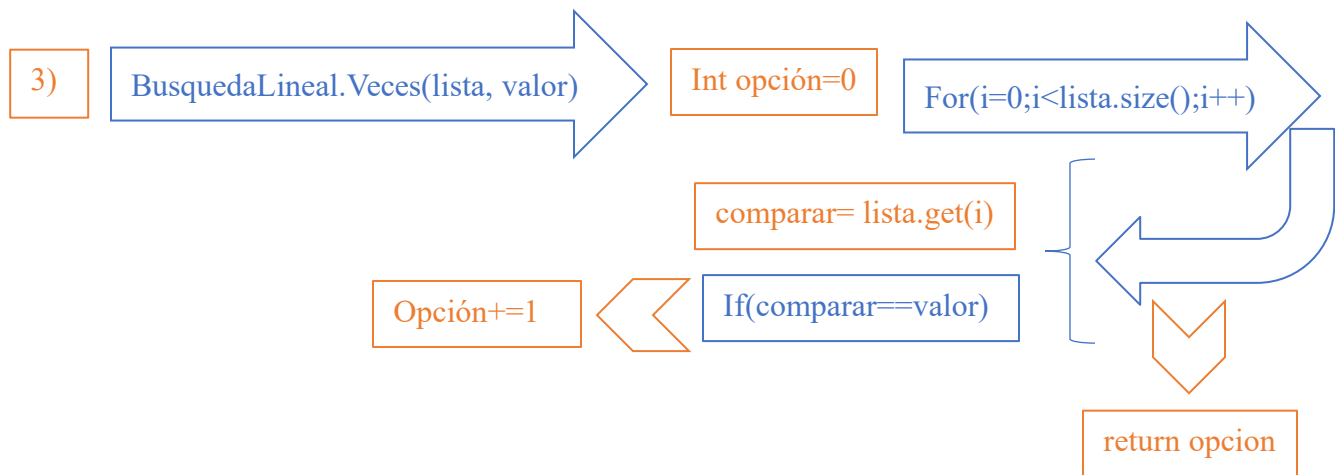
Guarda los valores que inserte el usuario dentro de la lista ligada, y posteriormente se manda la lista a la búsqueda lineal.



En este caso el booleano se inicializa como falso y solo cambia si el valor es igual a uno que este dentro de la lista y al final si se encuentra el elemento buscado, el valor de opcion se regresara y este será true en caso de que se haya encontrado o false en caso contrario.



Funciona igual que el booleano, solo que regresa la posición y en caso de que no se encuentre envía una posición que no se acepta en las listas.



En el ultimo caso se inicializa con cero para que en caso de que no se encuentre el valor, pues se devuelva cero y en caso de que se encuentre se le suma uno a la variable opcion que indica las veces que aparece el término.

- Relación con la teoría:

En las clases, los temas de búsqueda por medio de una búsqueda lineal hacen referencia a la búsqueda de objetos o elementos de forma natural, es decir comparando elemento a elemento y comprobando si cumple con los requisitos de lo que estamos buscando.

En este sentido el ejercicio consistió en la elaboración de la búsqueda de un elemento, comparando uno con uno los elementos que se encuentran en una lista con el elemento que se busca, y este método de búsqueda cuenta con tres versiones, con las que se puede determinar si se encuentra el término (Imagen 1). Para poder iterar en la lista, se utiliza un `for` y un `if` para poder comprobar que el elemento que se busca y el elemento de la posición sean los mismos y en caso de cumplirse se lleve a cabo la operación correspondiente para poder notificar que existe el elemento en la lista.

```
if(a==valorbuscado){  
    opcion=true;  
}
```

Imagen 1

Para el caso del retorno del booleano se considera que el termino no existe y si se encuentra el booleano se modifica y si no se encuentra nunca cambia la consideración de que no existía en un principio. En el caso de que se regresa la posición del término, se coloca una posición que no pertenece a la lista, y que en caso de que no se encuentre el valor esta no se modifica y no toma el valor del índice. Por ultimo para el caso de que se tenga que regresar el numero de veces que aparece un valor se crea un contador que se inicializa con un argumento que se envía desde el método principal, y posteriormente se modifica en el método en caso de que se encuentre el valor y lo regresa (Imagen 2).

```
for(int i=0;i<lista.size();i++){
    int a=lista.get(i);
    if(a==valorbuscado){
        b+=1;
    }
}
```

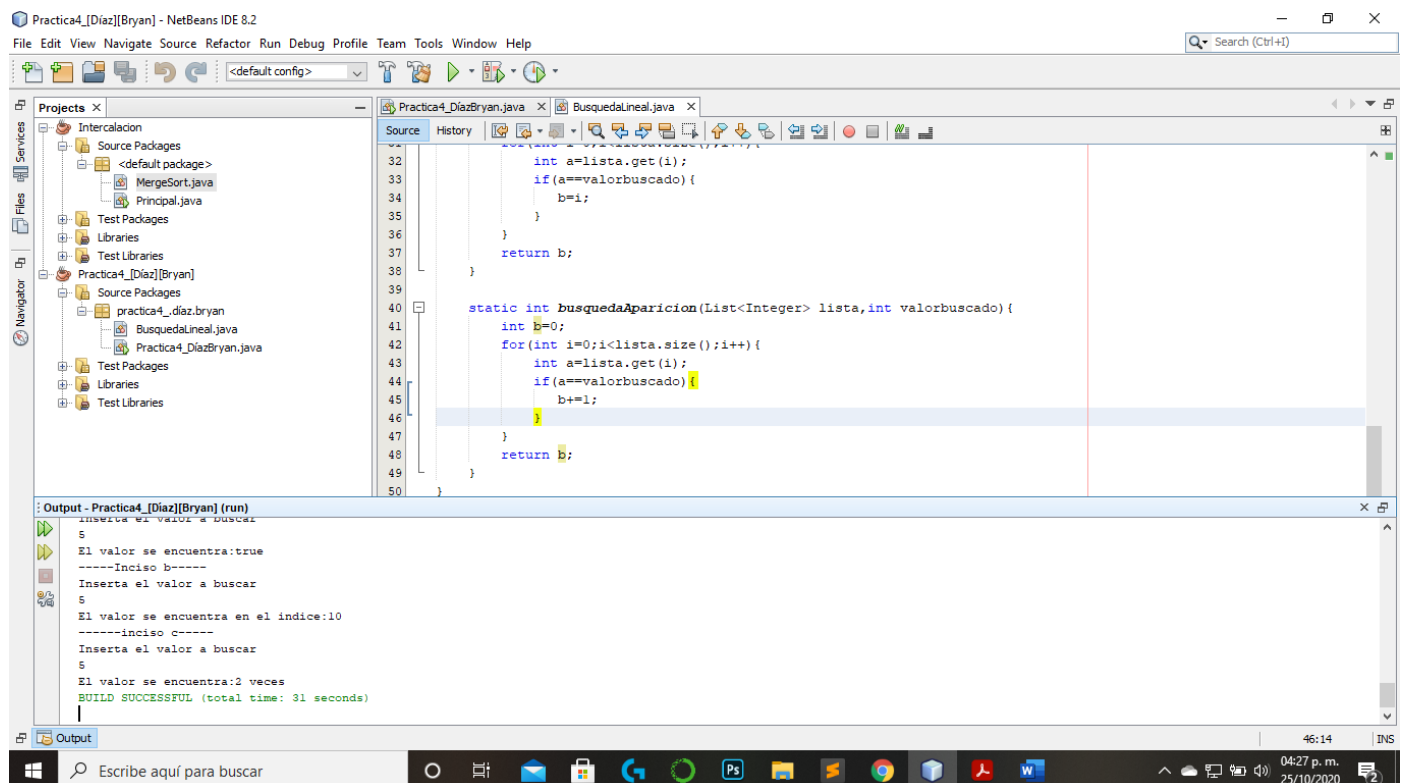
Imagen 2

De igual forma los métodos reciben parámetros, que son todos lo que declare, porque no vi necesario el crear una instancia sino que colocar métodos estáticos y poder acceder a ellos por medio de la clase, pero con respecto a estos métodos, en todos tuve que enviar la lista de la siguiente forma:

```
static int busquedaVeces(List<Integer> lista,int inicio,int fin, int valor,int contador){
```

Esto porque los parámetros necesitaban el tipo de lista que se recibía, y esto igual ayudo para el ultimo ejercicio de la práctica.

- Evidencia de implementación



- Ejercicio 3:

El tercer ejercicio consistía en la implementación de la búsqueda binaria, la cual se caracteriza por la búsqueda de un elemento, en una lista ordenada, y comenzando por el termino de en medio, aplicando el divide y vencerás ya que no busca elemento a elemento porque se considera un orden en la lista. El ejercicio requería el utilizar las dos variantes del algoritmo que consisten en notificar si existe con un booleano y el notificar las veces que se encontraba el elemento.

- Dificultades en el código

Dentro de la elaboración del ejercicio no tuve mucho problema con el método que regresaba un booleano, porque el código lo tenía apuntado en mis apuntes, así que solamente lo adapte, como mencionaba en el primer ejercicio lo mas importante era el mandar la lista por medio de parámetros y de esta forma el tener los métodos disponibles para poder modificar el estado de la lista y acceder a los valores de cada uno (Imagen 1).

En cuanto a esto en el segundo inciso se tiene que regresar el contador de las apariciones del valor a buscar, en este caso como son funciones recursivas si se declara un valor dentro de la función este se repite y no guarda el valor que compruebe las n veces que se encontraba el termino, era necesario el tener una variable global, pero para no crear un atributo, simplemente mande un valor como parámetro al método, y después este lo modificaba y lo enviaba de regreso (Imagen 2).

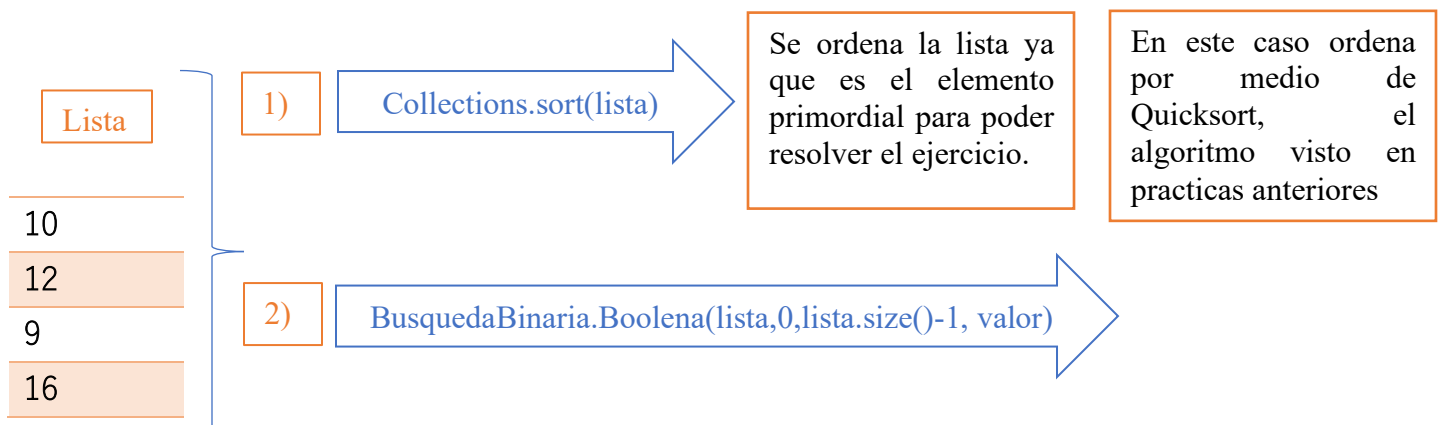
```
static int busquedaVeces(List<Integer> lista,int inicio,int fin, int valor,int contador){
```

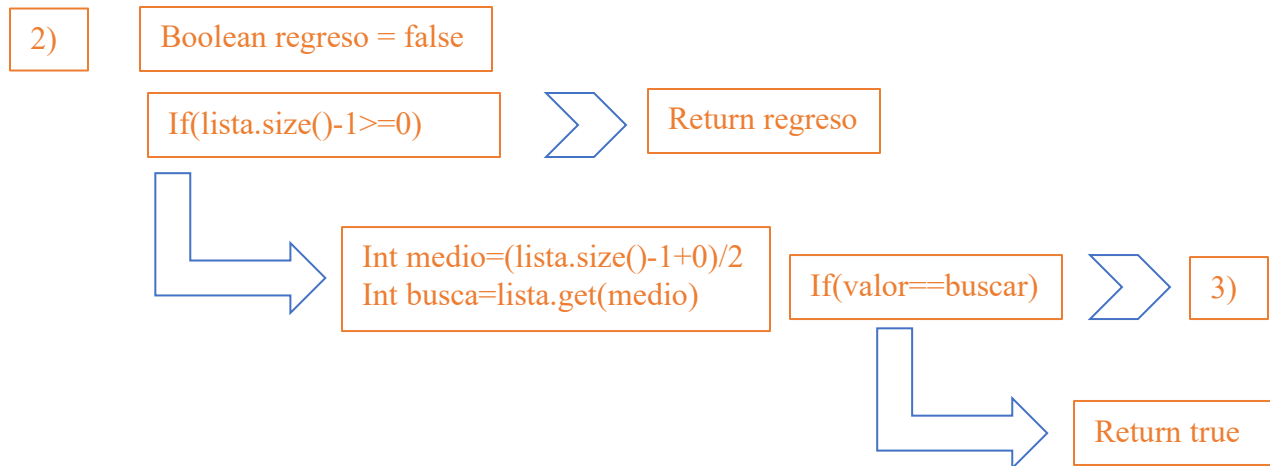
Imagen 1

```
static int busquedaVeces(List<Integer> lista,int inicio,int fin, int valor,int contador){  
    .....  
    return contador;
```

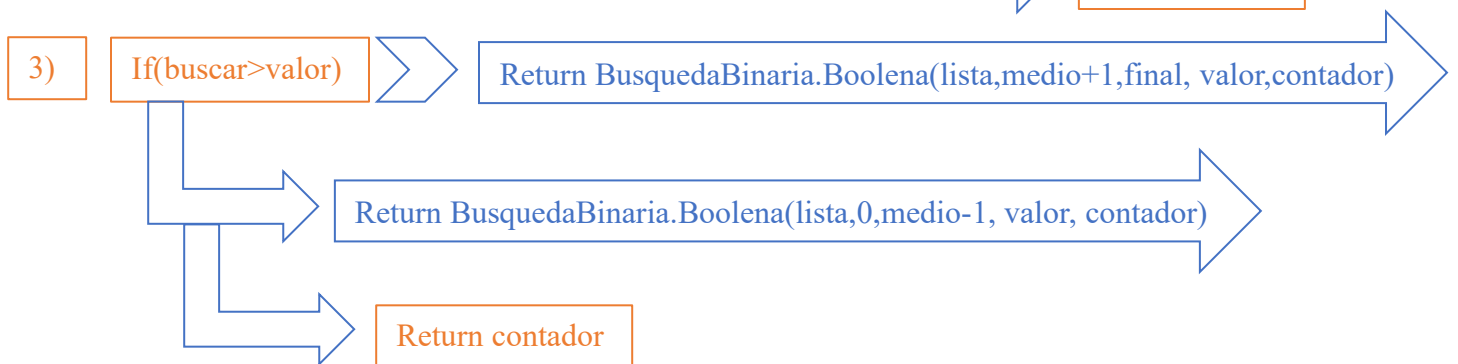
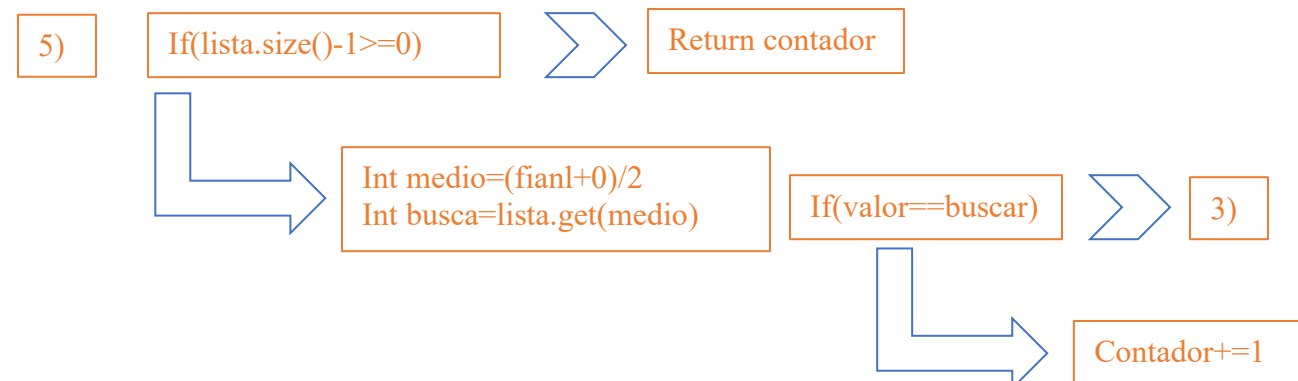
Imagen 2

- Diagrama de funcionamiento:





En este caso busca los valores y lo principal es que busca sobre una lista ordenada, por que al ser recursiva encuentra los valores sin problema de saltarse los valores, porque al momento del tercer if, esta operación si no se aplicara sobre una lista ordenada, podría saltarse elementos que no están en esa parte de la lista.



- Relación con la teoría:

En cuanto a la teoría que se ve dentro del ejercicio se tienen la búsqueda binaria, que es bastante completa, dentro de la primera condición que la lista esta ordenada y en segundo que el algoritmo comienza por la mitad de la lista, porque al estar ordenada las condiciones que permiten que el algoritmo se desplace sobre la lista asegurarían en estar iterando sobre posiciones que terminaran con encontrar el valor.

Entonces para ordenar la lista, se necesita un algoritmo de ordenamiento y en este caso funciono mi implementación, pero al ver que existe la clase Collections, y el método sort(), únicamente la importe y mande a ordenar la lista, esto fue por tratar de encontrar métodos y clases dentro de Java que pudieran hacer más eficiente el programa y en un futuro poder implementarlos (Imagen 1).

```
System.out.println("\nEjercicio 3\n----inciso a----");
System.out.println("Se ordena la lista");
Collections.sort(listaE2);
```

Imagen 1

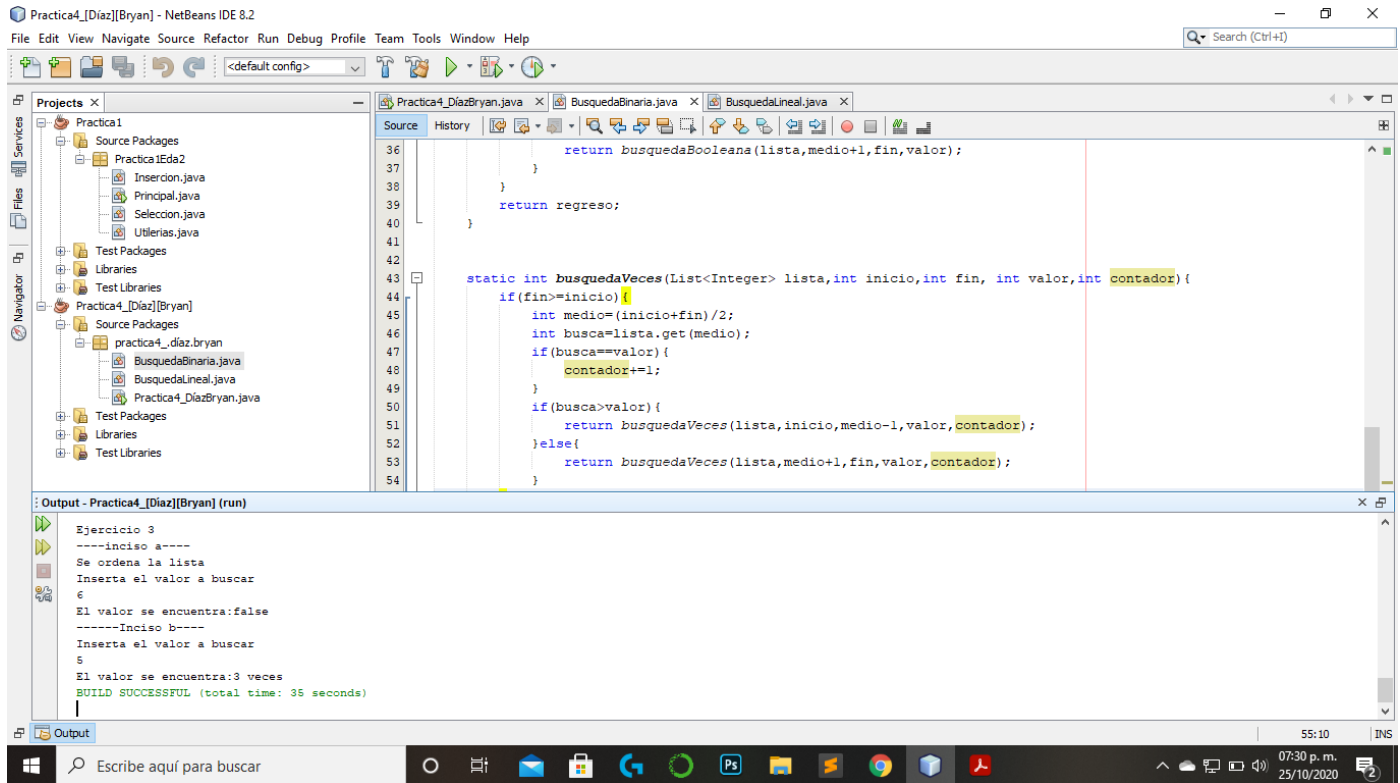
Como se mencionaba el método de búsqueda es recursivo por lo que se implementa como una QuickSort, pero este no ordena nada, solamente busca por lo que solo toma índices como referencia de la mitad de la lista y posteriormente comienza la recursividad mandando a llamarse el método, pero ahora con los índices que indique la condición simple if, de esta forma se va iterando en la lista por medio de índices y conforme al valor de los elementos que al estar ordenados aseguran el encontrar el valor si existe el valor (Imagen 2).

```
if(busca>valor){
    return busquedaBooleana(lista, inicio, medio-1, valor);
}else{
    return busquedaBooleana(lista, medio+1, fin, valor);
}
```

Imagen 2

Y con respecto a la carga teórica el último ejercicio es el verdadero reto, ya que es buscar dentro de una lista de objetos, y básicamente eso me trajo muchos problemas que resolver. Pero estos tres primeros ejercicios, fueron con la herramienta para poder resolver el cuarto y además que son de ayuda para poder pensar en la implementación de nuestro proyecto, por ello es por lo que el último ejercicio lo trate de desarrollar más.

- Evidencia de implementación



- Ejercicio 4:

El ultimo ejercicio requería el elaborar una clase Automóvil, en esta definir como mínimo tres atributos y tres métodos, donde lo mas importante serían los atributos ya que posteriormente se crearían 5 objetos en el método principal, y estos se añadirían a una lista del tipo Automóvil y posteriormente se buscarían los objetos por medio de su nombre y del modelo, donde estos dos atributos deberían ser fijo u obligatorios para poder proceder con la búsqueda de los objeto por medio de la búsqueda lineal y de la binaria.

- Dificultades en el código

Dentro de la creación de los métodos lo no tuve problemas para hacerlos, ya que he estado practicando poco a poco, pero lo que si se me dificulto fue el utilizar los métodos de búsqueda lineales y binarios. Lo que primero realice fue la definición de la clase por medio de los atributos y de los métodos, después cree los objetos, y al por ultimo procedí a modificar los algoritmos de búsqueda para poder encontrar los elementos, por nombre y por modelo.

Realice la búsqueda lineal sin mucho cambio en los métodos ya que en realidad al compararse valor a valor, lo que hice fue únicamente el obtener el nombre o el modelo, y compararlo, para esto utilice el método `get(posición).nombre` o `get(posición).modelo`, para poder obtener el valor del elemento en la posición, y guardar el valor por medio de `Add(get(Posición))` y obtener una lista con los valores que cumplan con la condición del nombre o del modelo (Imagen 1).

```

List<Automovil> listaRegreso =new LinkedList<>();
for(int i=0;i<lista.size();i++){
    int a=lista.get(i).modelo;
    if(a==valorbuscado){
        listaRegreso.add(lista.get(i));
    }
}

```

Imagen 1

Pero la parte mas compleja fue el implementar la búsqueda binaria, porque la premisa de que se tienen que tener ordenados los valores es muy importante, en este caso lo primero que hice fue ordenar los modelos, para no tener que lidiar con ello, ya que los algoritmos de ordenamiento normalmente se aplican a una lista o eso es lo que he visto y estoy seguro de que se pueden aplicar a objetos, pero es necesario el organizar en base a un criterio, ya sea el nombre o el modelo. Por ello al organizar los modelos, solo se tenía que organizar por medio del nombre.

Entonces la parte difícil fue el aplicar la búsqueda binaria al nombre, de ahí que investigue la parte de como descomponer una cadena en caracteres, y encontré el método charAt(posición), el cual toma el carácter que se indique en la posición y con este pude comenzar a comparar por el valor de Unit Code, de cada carácter, y mediante dos condicionantes de los primeros dos caracteres de las cadenas, se podía buscar el objeto por el nombre (Imagen 2).

```

int medio=(inicio+fin)/2;
char a=lista.get(medio).nombre.charAt(0);
char c=lista.get(medio).nombre.charAt(1);
char b=valor.charAt(0);
char d=valor.charAt(1);
if(a==b && c==d){
    listaRegresoNombre.add(lista.get(medio));
}

```

Imagen 2

Otra parte importante fue el como declarar las listas, porque estas se regresaban al método principal para poder verificar su contenido, y lo que sucede con los algoritmo recursivos es que las asignaciones se repiten y no se guardan valores si se hacen dentro de estos, y esta lista de retorno no debía reiniciarse, por lo que coloque como un método la creación de una instancia de tipo cola, de esta forma al ser estática podría ser vista por cualquier método y modificarse por cualquier elementos únicamente con el nombre, de ahí que en el método de búsqueda binaria por nombre.

Por ello se crearon dos listas del tipo automóvil donde se guardarían los valores y al final se enviarían al método principal, para poder ver el contenido de estas. Aunque al ser estáticas las creaciones de las listas estas podrían verse desde la clase principal, y desde ahí ver el contenido, pero como pide el retorno de las listas, se hizo una forma de hacerlo de tal manera que se tengan que retornar las listas.

```

static List<Automovil> listaRegresoModelo =new LinkedList<>();
static List<Automovil> listaRegresoNombre =new LinkedList<>();

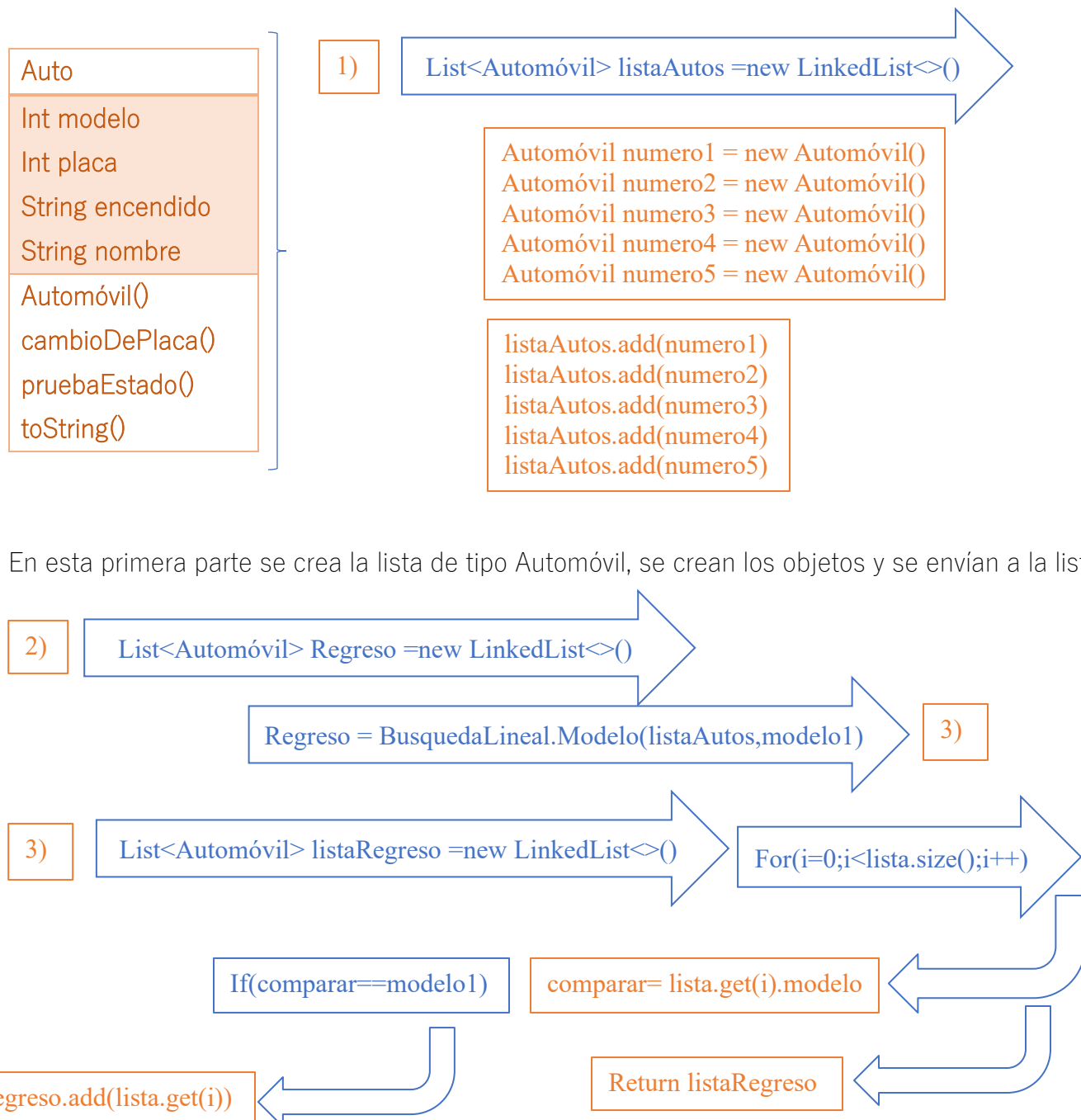
```

Algo no tan relevante pero importantes es la creación de la clase toString(), esto porque al mandar los elementos de la lista que se regresa al principal, se tienen que imprimir los valores de cada atributo para saber si cumplen con el elemento a buscar, y con el método toString() se realiza la impresión de los atributos del elemento que se guarde en las listas, por medio del get que se encuentra en el método imprimirListaA().(Imagen 3).

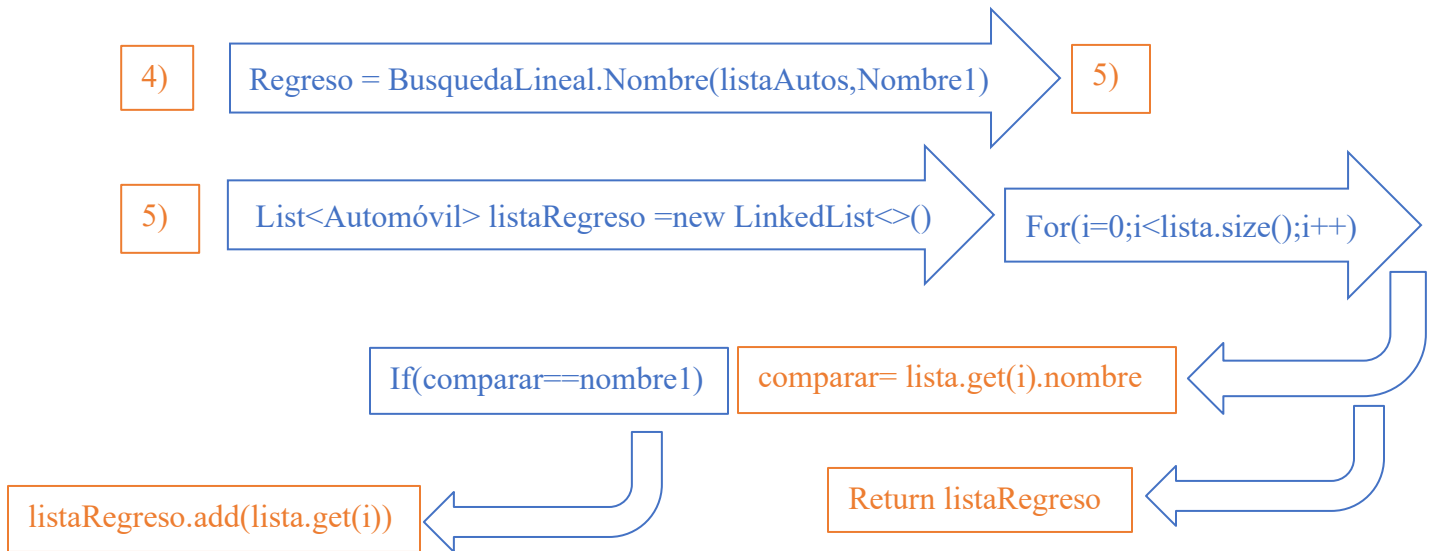
```
public String toString(){
    return "modelo:"+modelo+" nombre:"+nombre+" placa:"+placa+" encendido:"+encendido;
}
```

Imagen 3

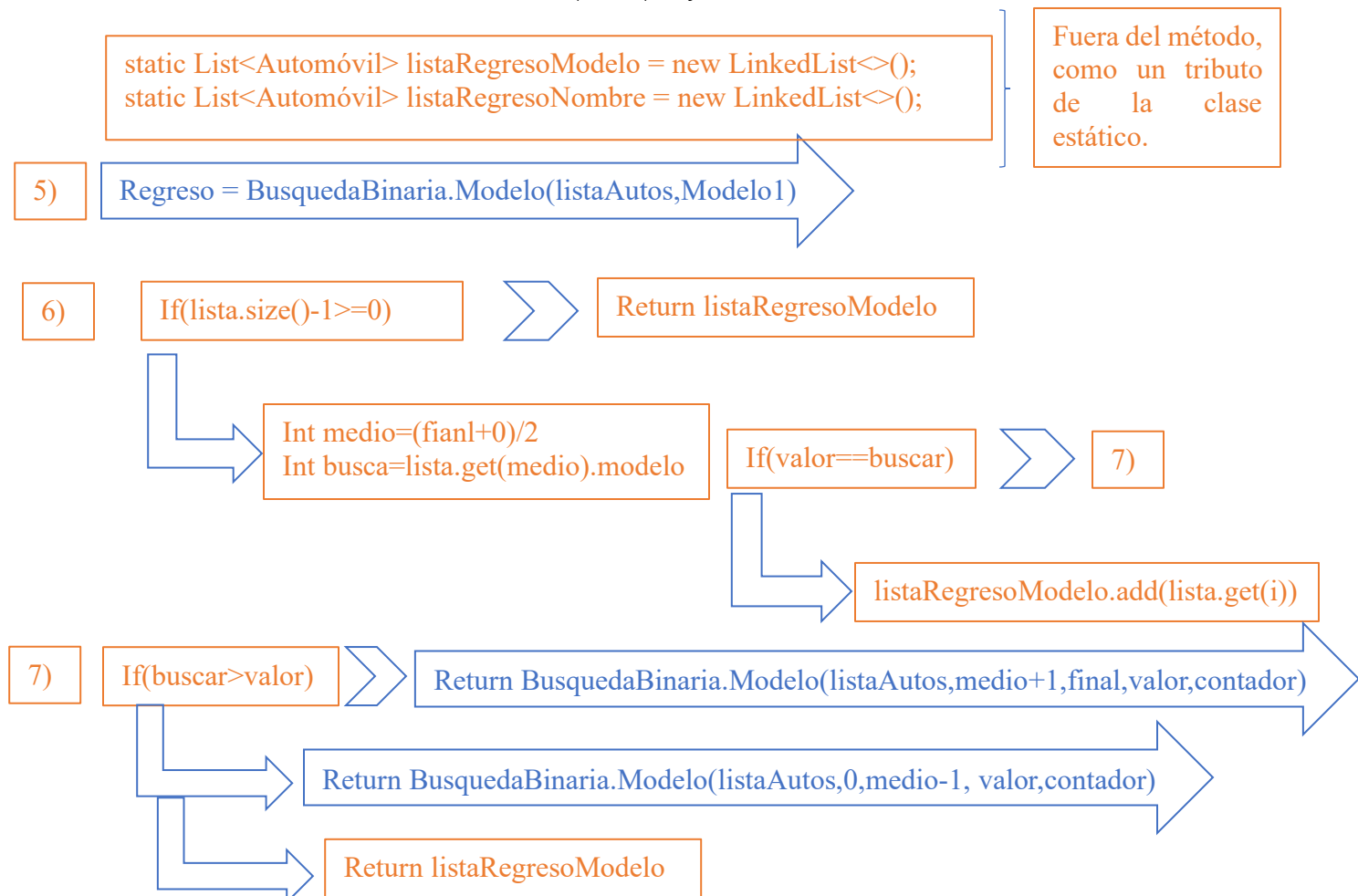
- Diagrama de funcionamiento:



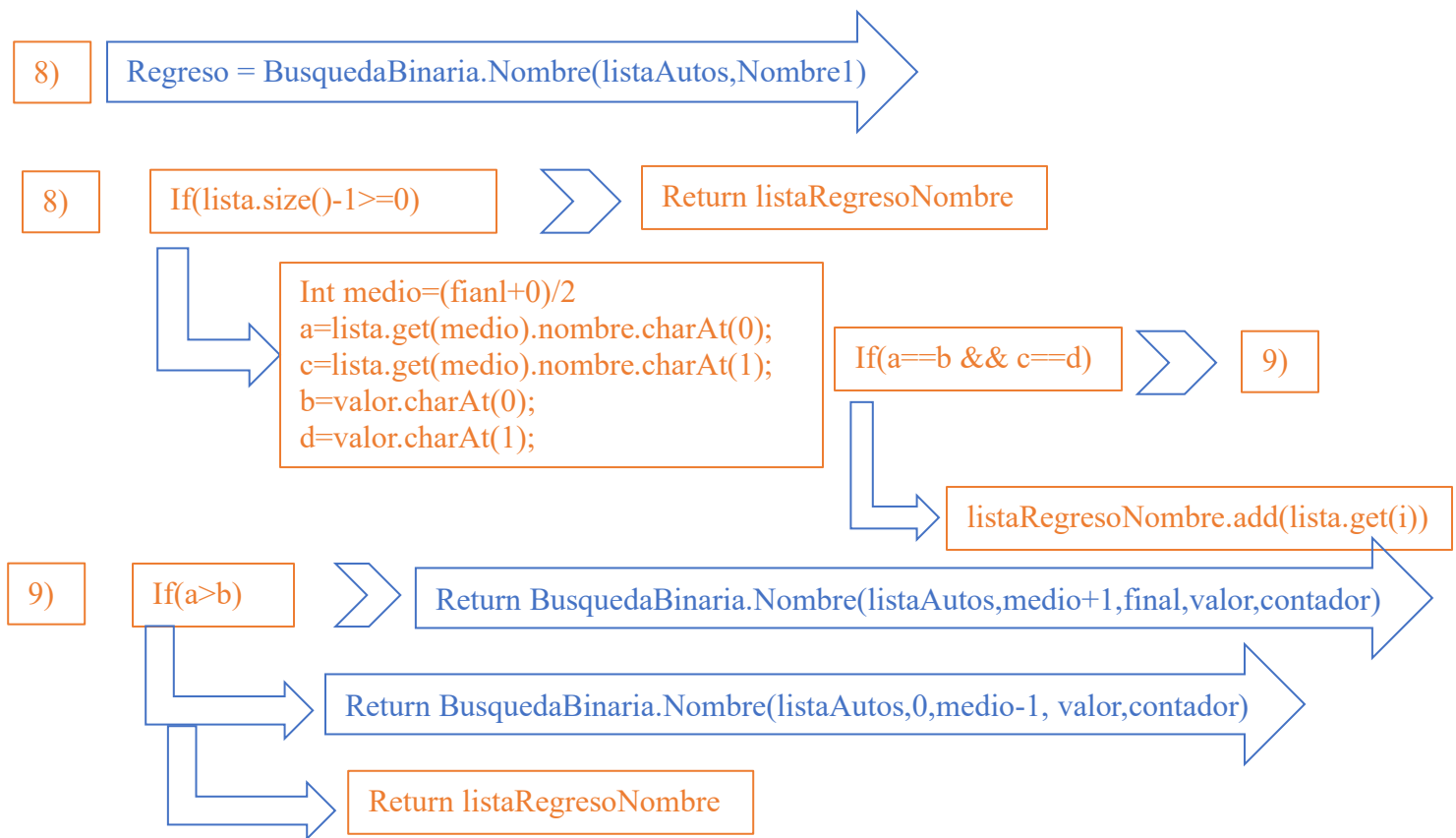
En este caso se crean dos listas, una que reciba los elementos que se encontraron y la segunda que se regresara, además que para obtener el modelo se tiene que utilizar los operadores (.) como en c que ayudaban a indicar los miembros, y en el caso que el miembro y el elemento a buscar, coincidan se agrega a la lista, y si se encuentra varias veces se almacena en la lista.



Algo importante es el retorno que tienen los métodos de búsqueda lineal porque estos tienen el tipo de dato List<Automóvil>, para que se pudiera devolver la lista, aunque se pudiera hacer estática y de esta forma acceder a ella desde el método principal y ver los valores.



Para el caso de la búsqueda binaria se tiene la condición de que se tiene que regresar una lista, pero a ser recursiva el valor de la lista al ser declarado pues se va reiniciando, y no se guardan los valores, por lo que decidí crear un atributo y colocarlo estático para que este guardara los valores y regresarlo al método principal, e igual se podría consultar directo desde el método principal al ser estático.



En este caso, coloque en orden el valor del modelo, y con el algoritmo de búsqueda, resultada sencillo encontrar por el modelo, pero el encontrar por medio del nombre resulta mas conflictivo ya que ordenan por medio de los caracteres del nombre generaría cambios en el primer ordenamiento, además que se tiene que considerar el segundo carácter para ordenar, por ello es que se obtiene n los dos primeros caracteres de cada cadena, y se comparan y en caso de que no coincidan estos dos con los otros, se divide la búsqueda, esto esta mal relativamente porque si se implementar por separado no hay problema con las búsquedas binarias pero si son en distinto orden los valores se terminan mezclando y alguno de las dos búsqueda no encontraría el valor y al otra sí.

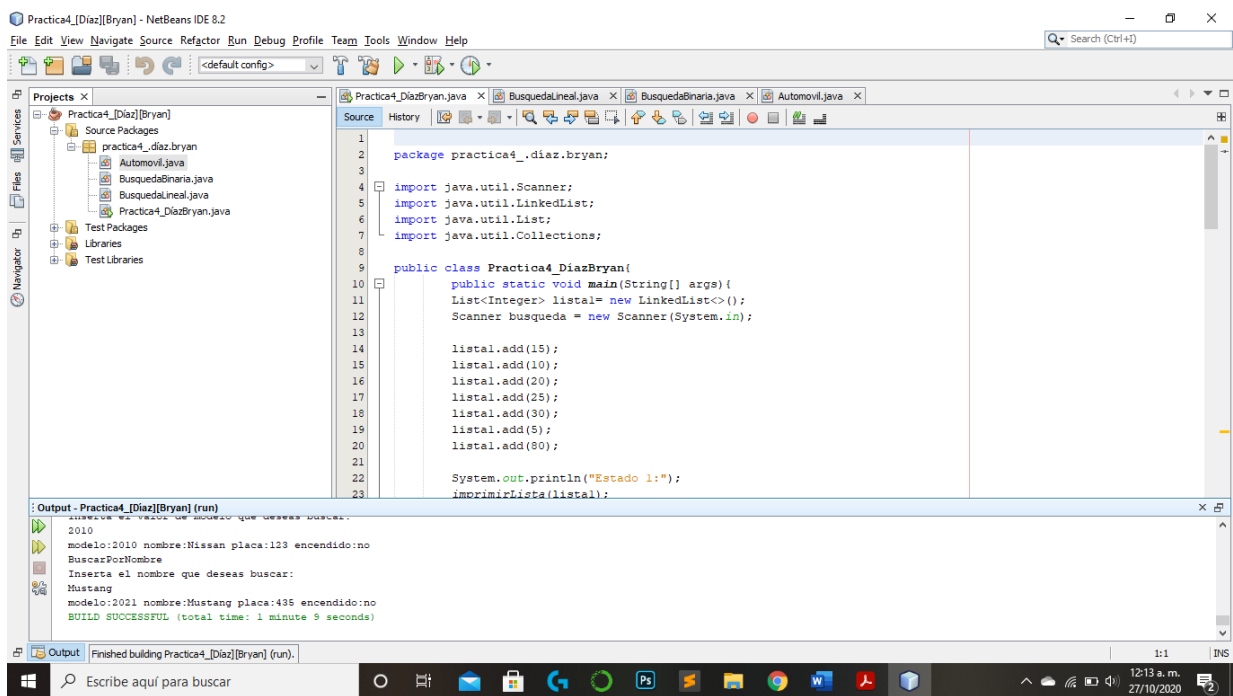
- Relación con la teoría:

La teoría detrás del ejercicio es bastante, porque si no se saben implementar las listas, simplemente no puedes realizar el ejercicio por medio de estas, además que no es solo eso sino que también se necesita el saber los tipos de búsqueda y saber cómo funcionan, de lo contrario no sería posible el saber que métodos de las listas utilizar y cuando utilizarlos en los algoritmos de búsqueda.

Por otro lado el tener las consideraciones necesarias para la búsqueda de los valores en la búsqueda binaria es el orden y sin este el algoritmo posiblemente jamás encontraría el valor dentro de la lista, además que al ser objetos con distintos atributos y posiblemente ordenables en base a un criterio que se aplique a cualquiera de los atributos, es conflictivo es buscar un punto del cual decidir para ordenar o si es mas conveniente el tomas las comparaciones o decisiones dentro del programa necesarias para que se pueda decidir.

Además de lo anterior, el ejercicio tiene un peso dentro de otra materia ya que si bien no es el mismo contenido temático estos se complementan, ya que al conocer Java por POO, se puede acelerar el proceso de entendimiento de los requisitos de los ejercicios y en la adecuada implementación de las clases y métodos para la solución mas precisa de cada ejercicio.

- Evidencia de implementación



```
1 package practica4_.diaz.bryan;
2
3
4 import java.util.Scanner;
5 import java.util.LinkedList;
6 import java.util.List;
7 import java.util.Collections;
8
9 public class Practica4_DiazBryan{
10     public static void main(String[] args){
11         List<Integer> listal= new LinkedList<>();
12         Scanner busqueda = new Scanner(System.in);
13
14         listal.add(15);
15         listal.add(10);
16         listal.add(20);
17         listal.add(25);
18         listal.add(30);
19         listal.add(5);
20         listal.add(80);
21
22         System.out.println("Estado 1:");
23         imprimirLista(listal);
24     }
25 }
```

Output - Practica4_DiazBryan (run)

```
Inserta el valor de donde se desea buscar:
2010
modelo:2010 nombre:Nissan placa:123 encendido:no
BuscarPorNombre
Inserta el nombre que deseas buscar:
Mustang
modelo:2021 nombre:Mustang placa:435 encendido:no
BUILD SUCCESSFUL (total time: 1 minute 9 seconds)
```

Conclusiones

En esta práctica aprendí bastante del lenguaje Java, mas que nada sobre las listas porque con anterioridad tenía la intención de implementarlas pero no sabía como hacerlo, así que esta practica me ayudo a poder implementarlas y a poder utilizarlas como medio de búsqueda de elementos y de objetos, donde los segundo resultado mas complejo en la búsqueda binaria pero se logro encontrar un forma de hacerlo, por ello puedo finalizar, escribiendo que entendí el funcionamiento de estos algoritmos de búsqueda y aprendí a utilizar las listas.