	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 -1317

Grupo: 10

No. de práctica(s): 4

Integrante(s): Mendoza Hernández Carlos Emiliano

No. de lista o brigada: 26

Semestre: 2023-1

Fecha de entrega: 29 de septiembre del 2022

Observaciones:

CALIFICACIÓN: _____

Práctica 4.

Algoritmos de búsqueda. Parte 1

OBJETIVOS

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

DESARROLLO

Ejercicio 1. Listas en Java

El manejo de listas en el lenguaje de programación Java se realiza a través de la interfaz List<E>, dicha interfaz cuenta con varias implementaciones. Las más utilizadas son ArrayList y LinkedList. Mediante la ayuda de NetBeans, crea un nuevo proyecto “Practica4_MendozaEmiliano”. En dicho proyecto agrega una clase principal y codifica el siguiente código.

```
package [...];

import java.util.LinkedList;
import java.util.List;

public class [...] {

    public static void main(String[] args) {
        List<Integer> lista1 = new LinkedList<>() ;

        lista1.add(15);
        [Aquí agrega 5 instrucciones más de "add" con los elements que quieras]
        lista1.add(80);

        System.out.println(" Estado 1 ");
        imprimirLista(lista1);
        System.out.println(" *** ");

        lista1.add(2,300);
        lista1.add(4,500);
        lista1.add(5,700);

        System.out.println(" Estado 2 ");
        imprimirLista(lista1);
        System.out.println(" *** ");

        lista1.set(1, 14);
        lista1.set(2, 16);
        lista1.set(7, 18);

        System.out.println(" Estado 3 ");
        imprimirLista(lista1);
        System.out.println(" *** ");

        List<Integer> lista2, lista3;
        lista2 = lista1.subList(2, 4);
        lista3 = lista1.subList(2, 4);
        imprimirLista(lista2);
        System.out.println(" *** ");
        imprimirLista(lista3);
        System.out.println(lista1.equals(lista2));

    }

    public static void imprimirLista(List<Integer> listaPrint){
        for(Integer var : listaPrint){
            System.out.println(var);
        }
    }
}
```

a. Explica la diferencia entre los métodos “set” y “add”.

Add es un método sobrecargado y existen dos variantes:

- ***add(E element)*:** Agrega el elemento que recibe como parámetro al final de la lista. Regresa un booleano.
- ***add(int index, E element):*** Agrega el elemento que recibe como parámetro en el índice especificado. No regresa nada.

Por otra parte, el método Set:

- ***set(int index, E element)*:** Reemplaza el elemento en la posición especificada de la lista por el elemento que recibe como parámetro. Además, regresa el método que estaba previamente en dicha posición.

Por lo tanto, la principal diferencia es que add agrega los elementos y set los reemplaza.

b. Explica el funcionamiento del método “sublist”.

subList(int fromIndex, int toIndex): Regresa una porción de la lista entre los índices *FromIndex* (inclusivo) y *toIndex* (exclusivo). Si *fromIndex* y *toIndex* son iguales regresa una lista vacía.

c. Investiga (y agrega las instrucciones respectivas en el programa) cuales son los métodos de Java para las siguientes operaciones

i. Borrar un elemento de la lista

Se utiliza el método Remove el cual está sobrecargado y tiene las siguientes variantes:

- ***remove()*:** Elimina el primer elemento de la lista y lo regresa como valor de retorno.
- ***remove(int index)*:** Elimina el elemento en el índice especificado y lo regresa como valor de retorno.

ii. Para saber si es vacía

***isEmpty()*:** Regresa *true* si la lista no contiene ningún elemento.

iii. Para buscar algún elemento

Para saber si el elemento existe en la lista:

***contains(Object o)*:** Regresa *true* si la lista contiene el objeto especificado.

Ejercicio 2. Búsqueda lineal

Agrega una nueva clase al proyecto llamada *BusquedaLineal*, en ella implementa 3 métodos para las 3 variantes de búsqueda lineal:

a) Un método que devuelva verdadero si un valor (clave) se encuentra en la lista y falso en caso contrario.
b) Un método que devuelva el índice donde se encuentra la clave.
c) Un método que devuelva el número de veces que aparece la clave en la lista.
   *Deberás implementar los algoritmos.

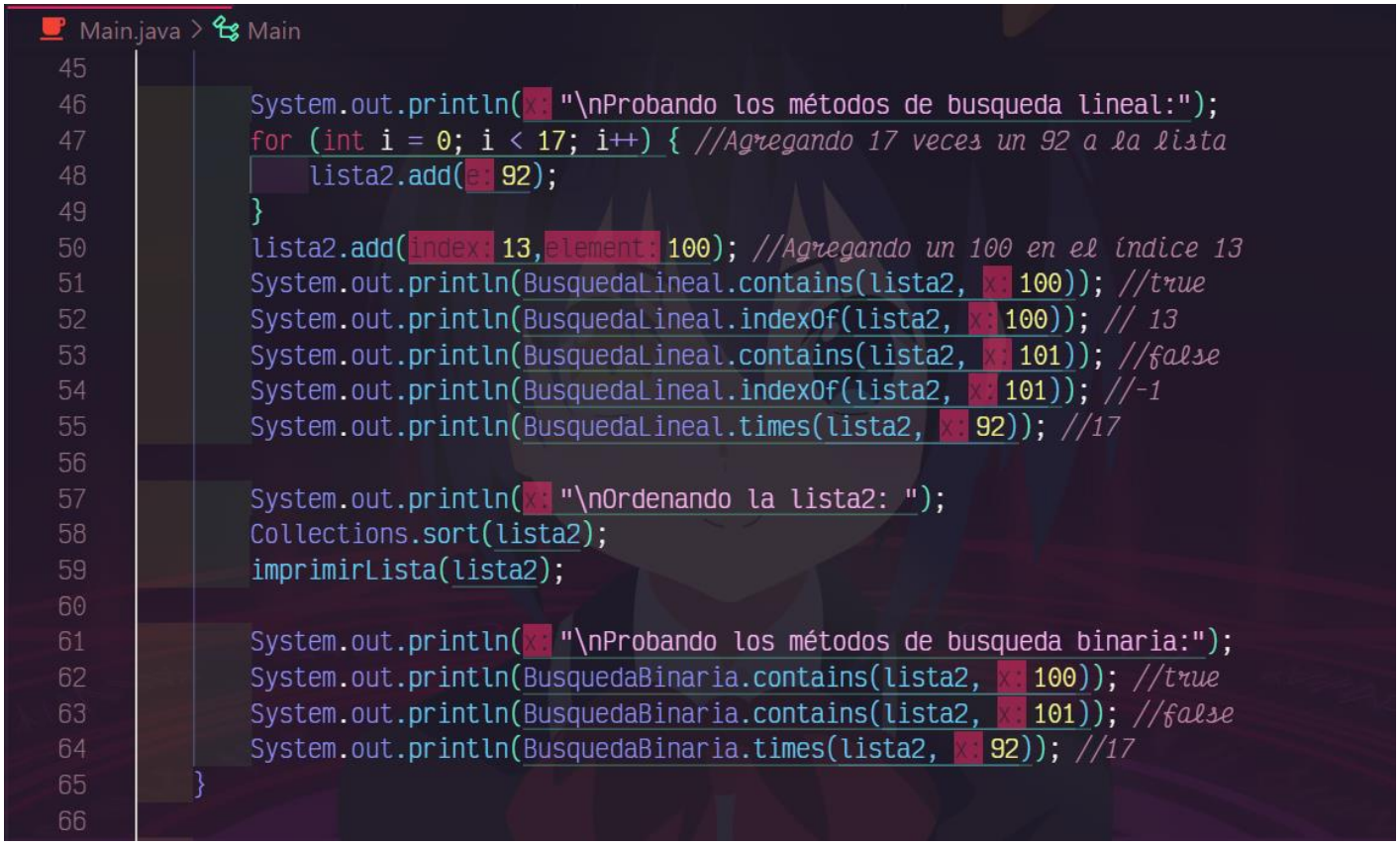
Ejercicio 3. Búsqueda binaria

Agrega una nueva clase al proyecto llamada *BusquedaBinaria*, en ella implementa las siguientes versiones de búsqueda binaria:

a) Un método que devuelva verdadero si la clave se encuentra en la lista y falso en caso contrario.
b) Un método que devuelva el número de veces que la clave se encuentra en la lista.

Comprueba el funcionamiento de tus métodos en la clase principal.

Probando los métodos en la clase *Main*

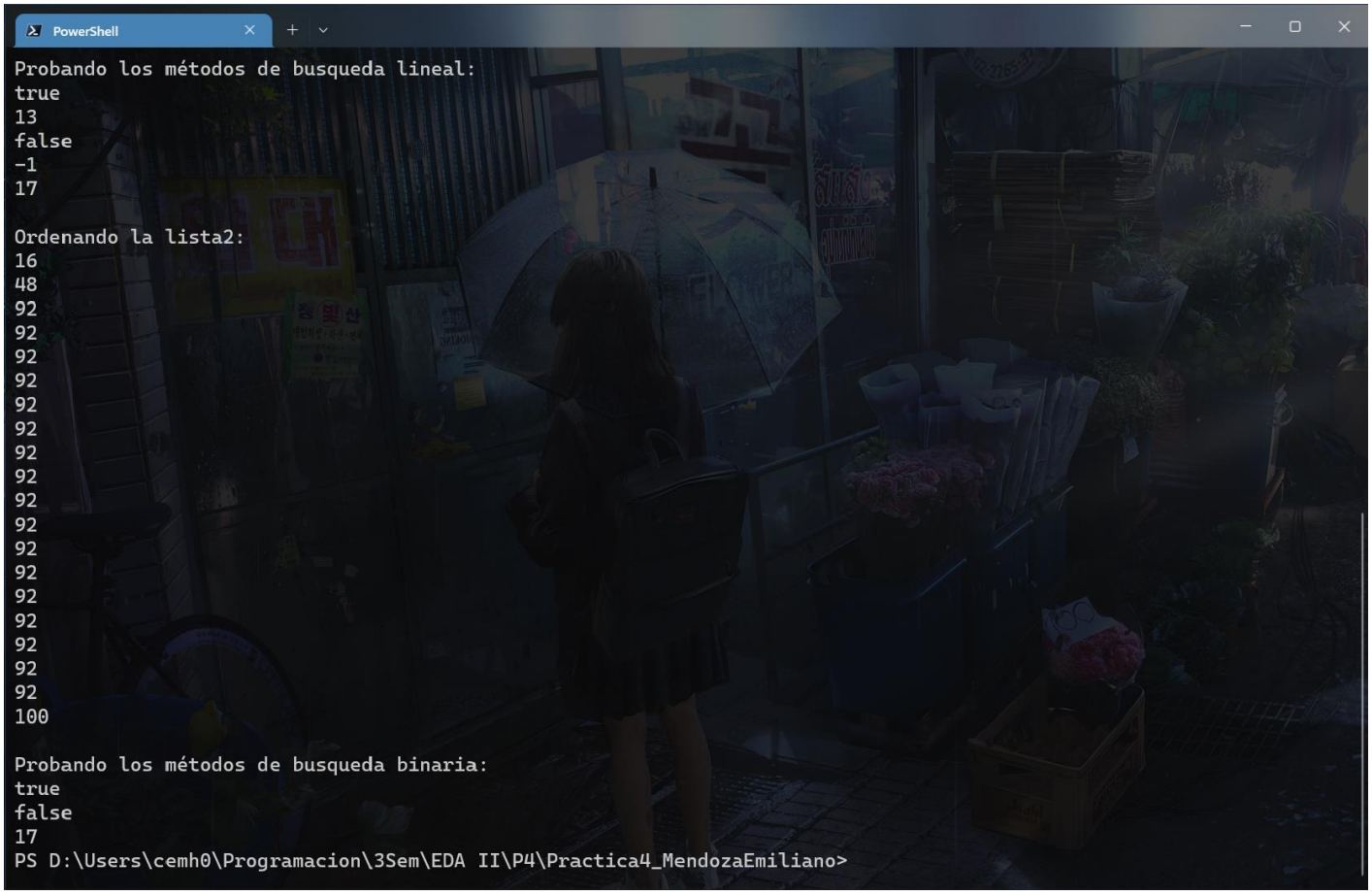


***Observaciones*:** Se utilizó la *lista2* para probar los métodos, la cual previamente solo tenía los números 16 y 48.

Para probar el método *times* (devuelve la cantidad de coincidencias del elemento en la lista), se agregaron 17 números 92. Se espera que al usar el método buscando números 92, devuelva un 17.

Para probar el método *contains* (devuelve *true* si el valor está en la lista) y el método *indexOf* (devuelve el índice de la posición en la lista donde se encuentra) se agregó un 100 en la posición 13 de *lista2*. Se buscó el número 100 con ambos métodos y se espera que devuelva un *true* y un 13. Luego, se buscó el número 101 (que no está en la lista) y se espera que devuelva un *false* y un -1.

Capturas de la ejecución de los métodos



***Nota*:** Se tuvo que ordenar la lista para poder usar la búsqueda lineal.

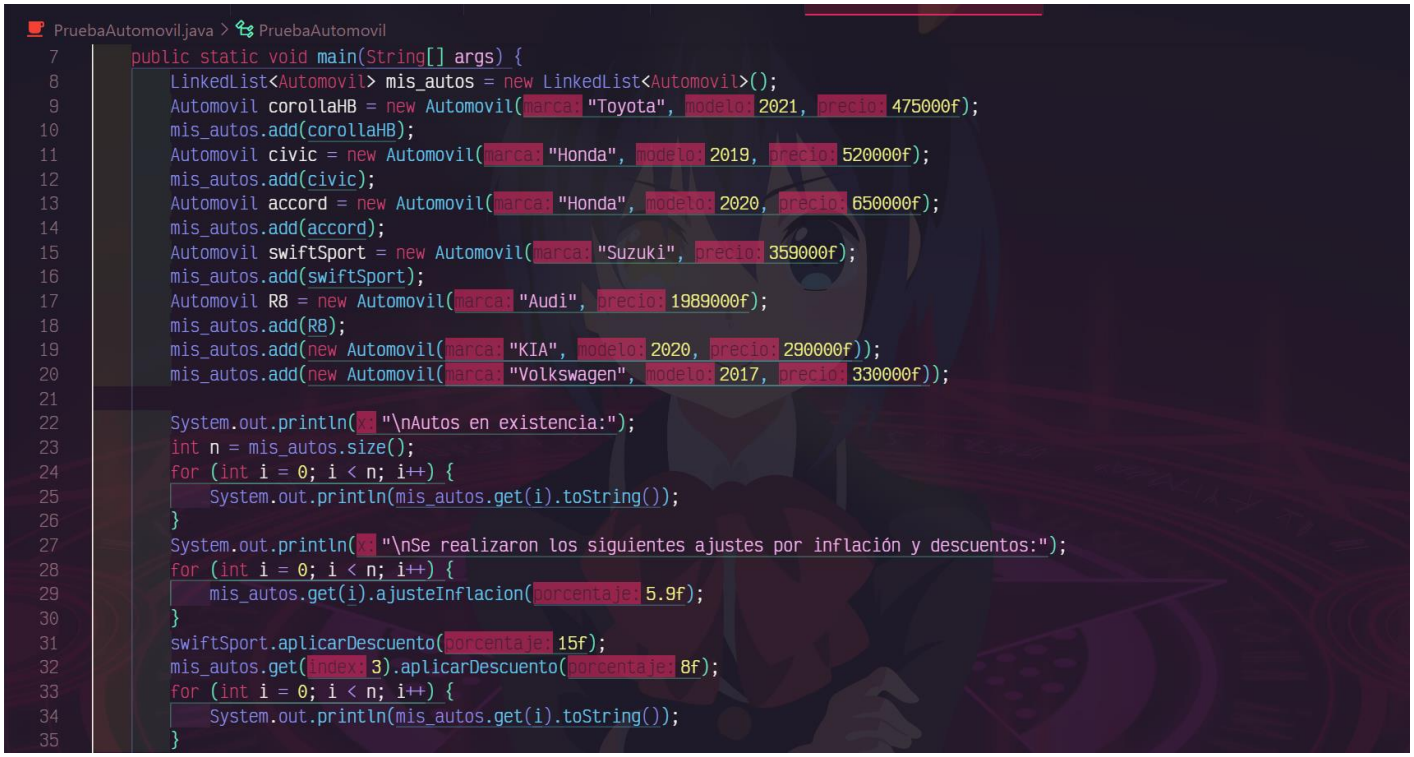
Ejercicio 4. Búsqueda en listas de objetos

a) Agrega una nueva clase al proyecto donde definas los atributos y métodos de un Objeto para crear el tipo de dato Automóvil (el diseño es libre, pero al menos debe tener 3 atributos, dos métodos, constructores diferentes y dos métodos adicionales de la clase).
b) Agrega en la clase de búsqueda lineal, dos métodos nuevos (puede ser basados en los métodos anteriores), con la diferencia que en lugar de devolver como parámetro “Integer”, devuelvan objetos y que reciban los siguientes parámetros:
    1) Marca (búsqueda por String)
    2) Modelo (búsqueda por int)

    La búsqueda deberá devolver una lista de apariciones de los elementos.

4.2 Agrega una nueva clase para este ejercicio y en ella crea una lista de Automóviles y comprueba el funcionamiento de los métodos realizados para las búsquedas.

Creando objetos de la clase *Automovil*



```java
PruebaAutomovil.java > PruebaAutomovil
    public static void main(String[] args) {
        LinkedList<Automovil> mis_autos = new LinkedList<Automovil>();
        Automovil corollaHB = new Automovil(marca: "Toyota", modelo: 2021, precio: 475000f);
        mis_autos.add(corollaHB);
        Automovil civic = new Automovil(marca: "Honda", modelo: 2019, precio: 520000f);
        mis_autos.add(civic);
        Automovil accord = new Automovil(marca: "Honda", modelo: 2020, precio: 650000f);
        mis_autos.add(accord);
        Automovil swiftSport = new Automovil(marca: "Suzuki", precio: 359000f);
        mis_autos.add(swiftSport);
        Automovil R8 = new Automovil(marca: "Audi", precio: 1989000f);
        mis_autos.add(R8);
        mis_autos.add(new Automovil(marca: "KIA", modelo: 2020, precio: 290000f));
        mis_autos.add(new Automovil(marca: "Volkswagen", modelo: 2017, precio: 330000f));

        System.out.println(x: "\nAutos en existencia:");
        int n = mis_autos.size();
        for (int i = 0; i < n; i++) {
            System.out.println(mis_autos.get(i).toString());
        }
        System.out.println(x: "\nSe realizaron los siguientes ajustes por inflación y descuentos:");
        for (int i = 0; i < n; i++) {
            mis_autos.get(i).ajusteInflacion(porcentaje: 5.9f);
        }
        swiftSport.aplicarDescuento(porcentaje: 15f);
        mis_autos.get(index: 3).aplicarDescuento(porcentaje: 8f);
        for (int i = 0; i < n; i++) {
            System.out.println(mis_autos.get(i).toString());
        }
```

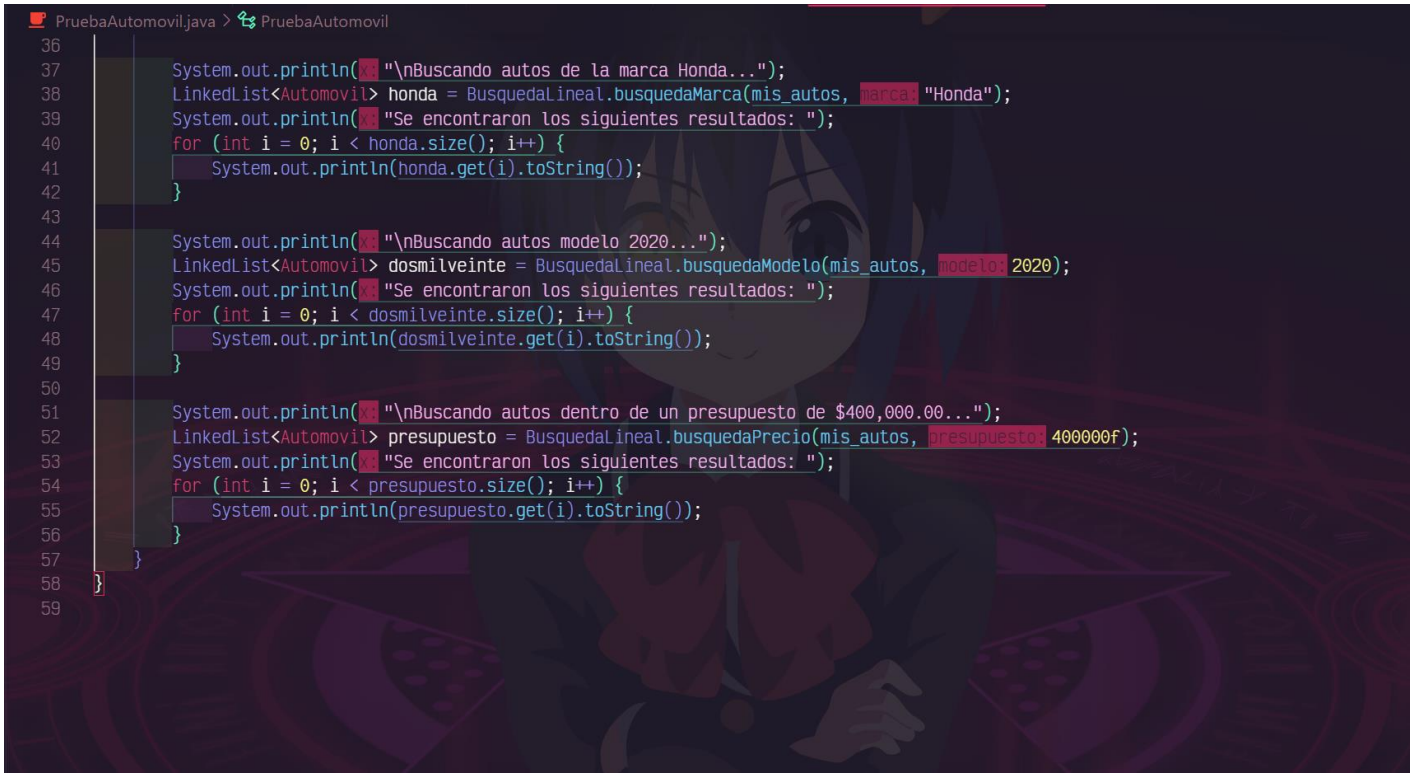
***Observaciones*:** Se probaron los dos constructores, el que tiene todos los parámetros y el que coloca el año más reciente por defecto. Además, los automóviles se agregaron a la lista ligada de dos maneras:

1) Instanciando el objeto y agregando posteriormente la instancia a la lista.
2) Instanciar directamente el objeto en la nueva posición de la lista.

La ventaja de la primera forma de agregar los autos a la lista es que podemos referirnos a la instancia explícitamente por un nombre (*civic*, *accord*, etc.). En cambio, de la segunda manera, debemos referirnos al objeto implícitamente a través de su índice en la lista (*autos.get(i)*).

También se probaron los demás métodos de la clase.

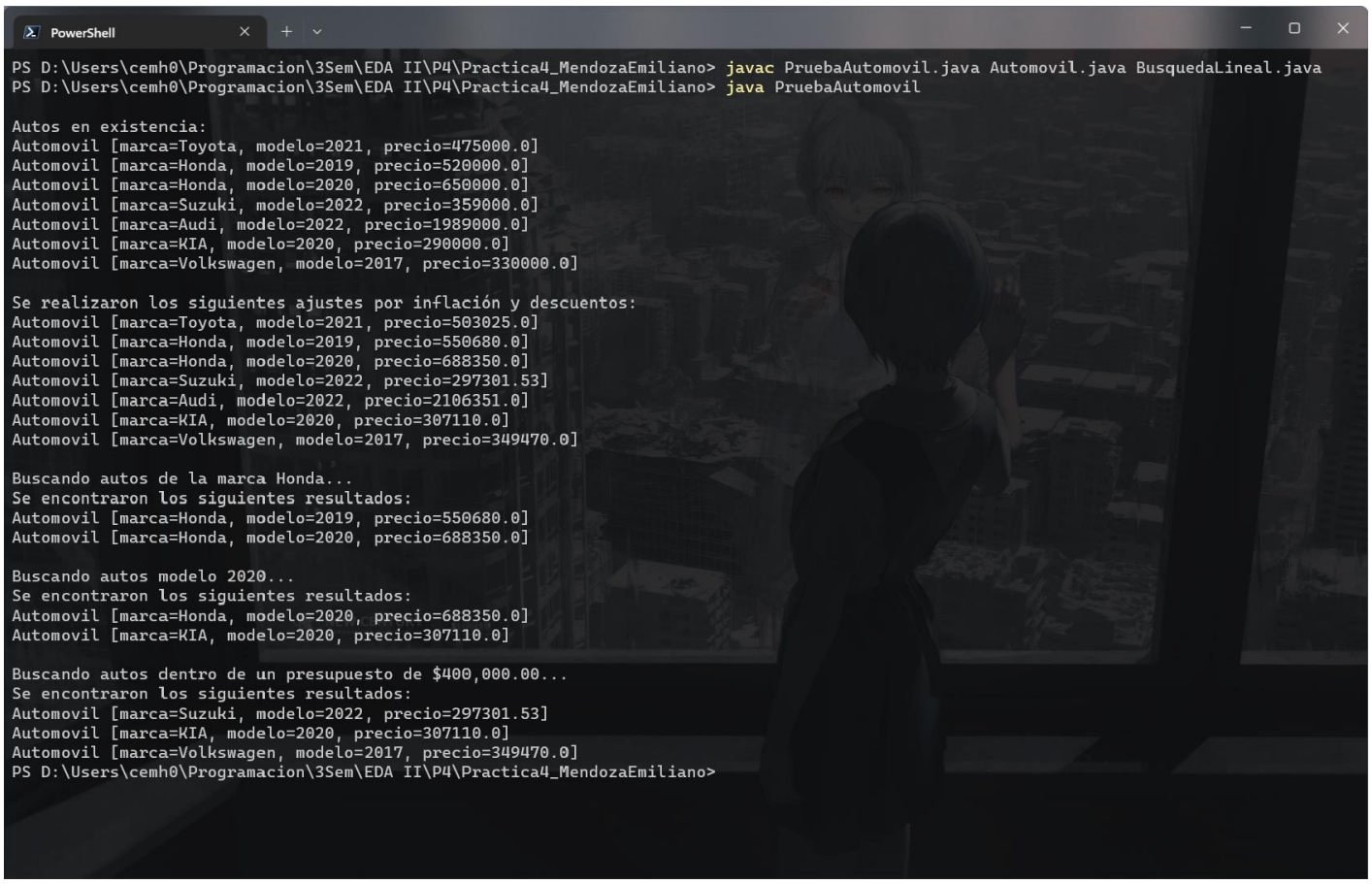
Probando los métodos en la clase *PruebaAutomovil*



```java
PruebaAutomovil.java > PruebaAutomovil
36
37          System.out.println( x: "\nBuscando autos de la marca Honda...");
38          LinkedList<Automovil> honda = BusquedaLineal.busquedaMarca(mis_autos, marca: "Honda");
39          System.out.println( x: "Se encontraron los siguientes resultados: ");
40          for (int i = 0; i < honda.size(); i++) {
41              System.out.println(honda.get(i).toString());
42          }
43
44          System.out.println( x: "\nBuscando autos modelo 2020...");
45          LinkedList<Automovil> dosmilveinte = BusquedaLineal.busquedaModelo(mis_autos, modelo: 2020);
46          System.out.println( x: "Se encontraron los siguientes resultados: ");
47          for (int i = 0; i < dosmilveinte.size(); i++) {
48              System.out.println(dosmilveinte.get(i).toString());
49          }
50
51          System.out.println( x: "\nBuscando autos dentro de un presupuesto de $400,000.00...");
52          LinkedList<Automovil> presupuesto = BusquedaLineal.busquedaPrecio(mis_autos, presupuesto: 400000f);
53          System.out.println( x: "Se encontraron los siguientes resultados: ");
54          for (int i = 0; i < presupuesto.size(); i++) {
55              System.out.println(presupuesto.get(i).toString());
56          }
57      }
58  }
59
```

***Nota*:** Se agregó un tercer método (*busquedaPrecios*) que usa la búsqueda lineal para encontrar automóviles menores a un límite de precio establecido (presupuesto).

Captura de la ejecución de los métodos



CONCLUSIONES

Algunos puntos que se concluyen de la primera actividad son los siguientes:

- El uso de listas en Java facilita y simplifica su implementación y operaciones.
- La API cuenta con bastantes clases predefinidas y métodos sobrecargados que podemos utilizar muy fácilmente.

Para el segundo ejercicio se puede concluir que:

- La búsqueda lineal es un algoritmo sencillo que se basa en la comparación de claves.
- Cada una de sus variantes funciona de manera similar, pero tienen diferentes propósitos.

Además, al realizar el ejercicio 3 se llegó a las siguientes conclusiones:

- La búsqueda binaria es más eficiente en cuanto a tiempos de ejecución porque no revisa la lista elemento a elemento, sino que en cada iteración divide la lista a la mitad.
- Una desventaja de la búsqueda binaria es que la lista debe estar previamente ordenada, lo cual conlleva al uso de un algoritmo de ordenamiento.

De la práctica 4 se concluye que:

- Es posible hacer la búsqueda lineal en colecciones con tipos de datos abstractos (*Automovil*), además.
- Se puede implementar búsqueda lineal buscando otros tipos de datos como *String* y *float*.

Dicho todo lo anterior, considero que se ha cumplido con cada uno de los objetivos planteados para el desarrollo de esta práctica dado que se identificó y comprendió el comportamiento de los algoritmos de búsqueda por comparación de llaves al codificar y realizar los ejercicios.

Como comentario final, considero que para esta práctica fue muy importante aplicar conocimientos de la programación orientada a objetos para implementar la solución al último ejercicio.

.