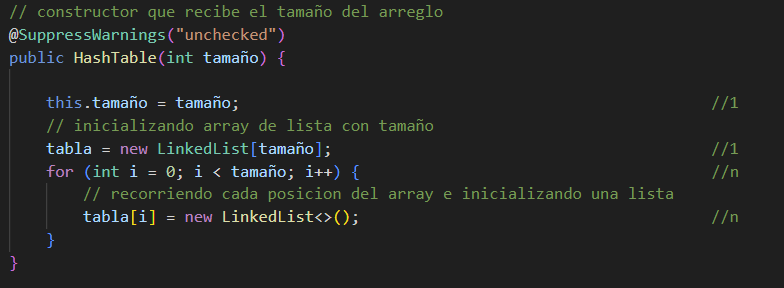
COMPLEJIDADES DE ALGORITMOS

**Clase HashTable**

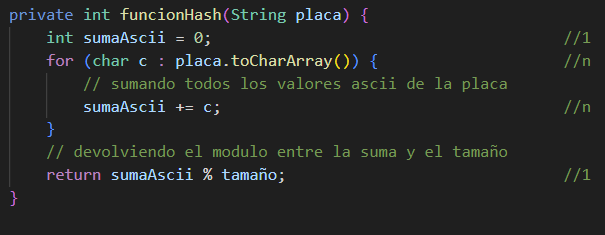
* Método:



Justificación de complejidad:

Este método tiene la complejidad O(n), porque itera una n cantidad de veces según sea el numero recibido por parámetro.

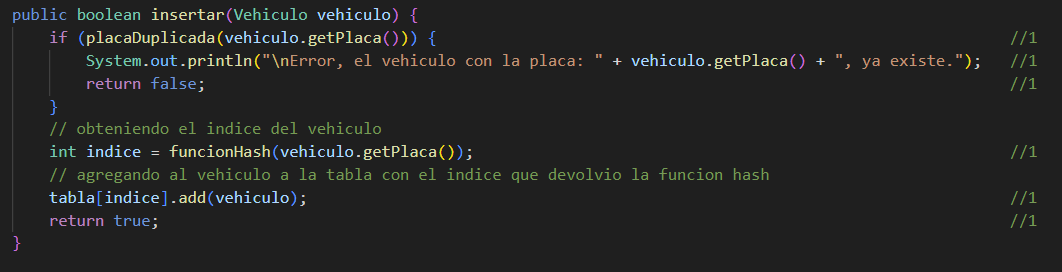
* Método:



Justificación de complejidad:

Este metodo tiene la complejidad O(n) ya que recorre un String n veces según sea el tamaño de la placa del vehiculo que se esta recibiendo como parámetro.

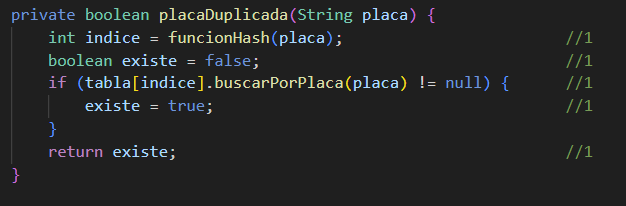
* Método:



Justificación de complejidad:

Este metodo tiene la complejidad O(1) ya que todas las acciones que realiza no dependen de un tamaño de entrada, y se ejecutan una sola vez.

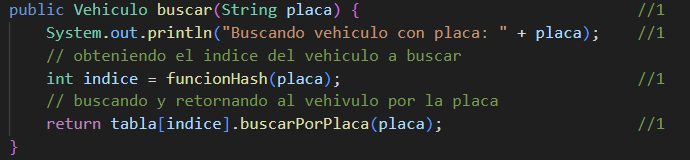
* Método:



Justificación de complejidad:

Este método tiene la complejidad O(1) ya que realiza una única búsqueda dada una placa recibida para verificar que no este duplicada en el sistema por parámetro, en la llamada del metodo.

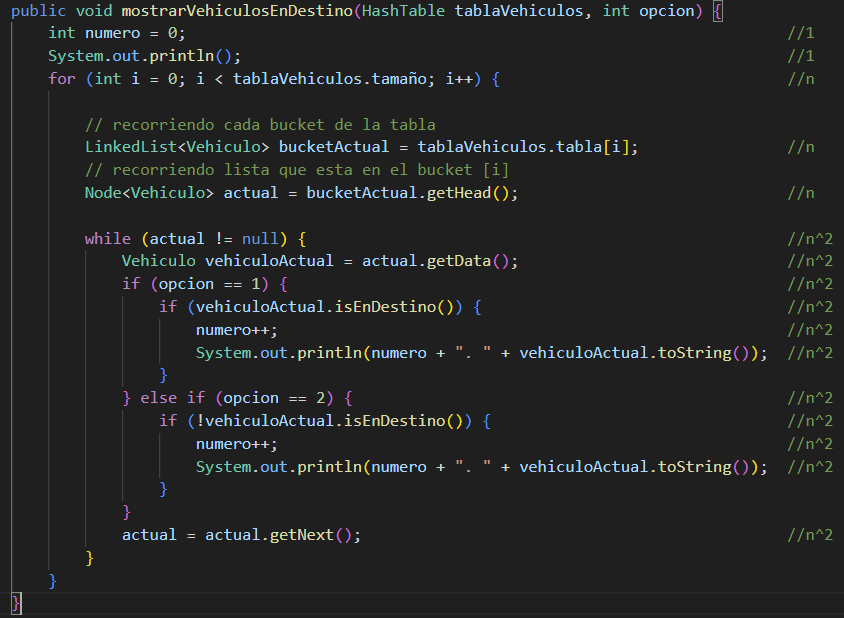
* Método:



Justificación de complejidad:

Este método tiene la complejidad O(1) ya que realiza una única búsqueda dada una placa recibida por parámetro, en la llamada del metodo.

* Método:

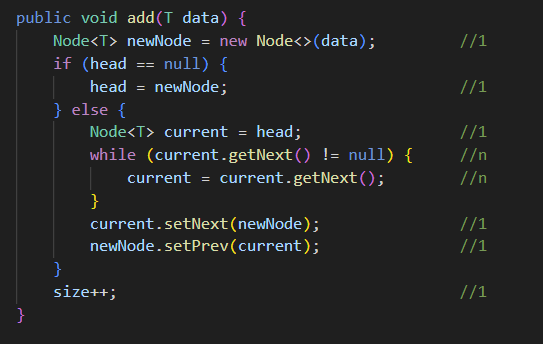


Justificación de complejidad:

Este metodo tiene una complejidad de O(n^2) ya que recorre un arreglo de listas enlazadas, y en cada posición del arreglo o cada lista, recorre esa lista n cantidad de veces para mostrar cada vehiculo en cada posición de cada lista.

**Clase LinkedList**

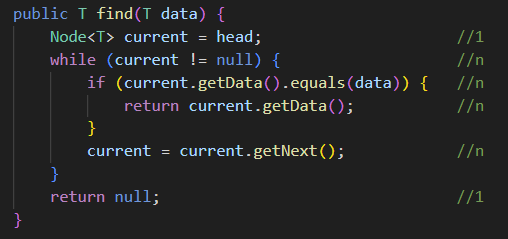
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n) ya que recorre los nodos siguientes de un nodo una cantidad de n veces hasta que el nodo ya no tenga un siguiente para colocárselo como siguiente y aumentar el tamaño de la lista.

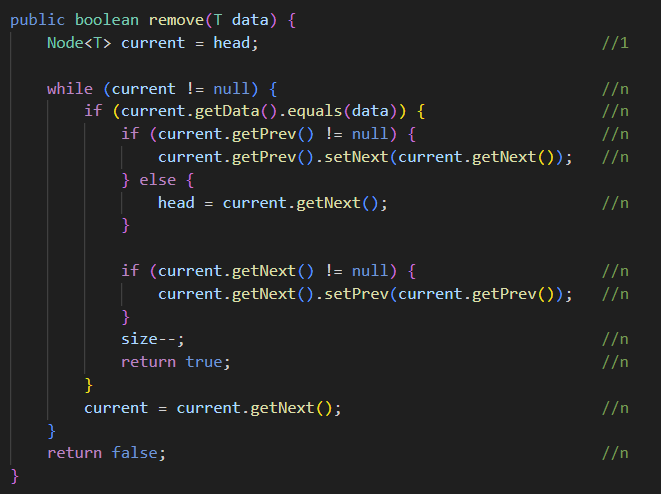
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n) ya que recorre un while n veces hasta que el nodo actual ya no tenga siguiente, o hasta que el dato que esta buscando sea igual al dato que se recibe por parámetro.

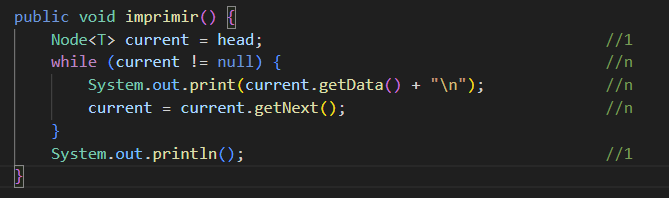
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n) ya que recorre un while n veces hasta que el nodo actual ya no tenga siguiente, o hasta que el dato que esta buscando sea igual al dato que se recibe por parámetro y lo elimina de la lista.

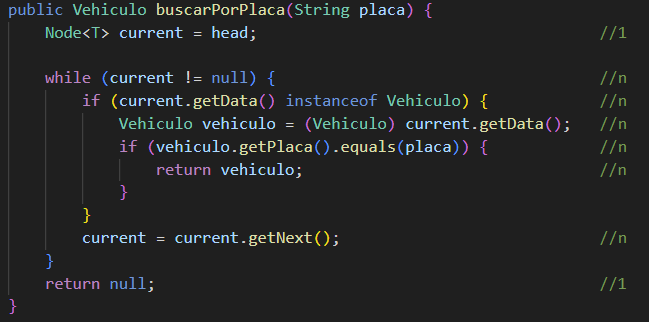
* Método:



Justificación de complejidad:

Este método tiene una complejidad de O(n) ya que recorre un while n veces hasta que el nodo actual ya no tenga siguiente y muestra cada objeto en consola.

* Método:

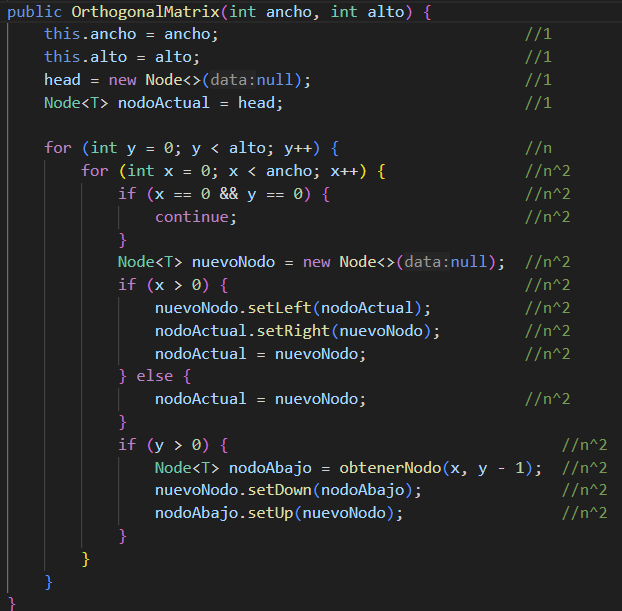


Justificación de complejidad:

Este metodo tiene una complejidad de O(n) ya que recorre un while n veces hasta que el nodo actual ya no tenga siguiente, o hasta que el dato que esta buscando sea igual al dato que se recibe por parámetro usando como metodo de comparación la placa.

Clase OrthogonalMatrix

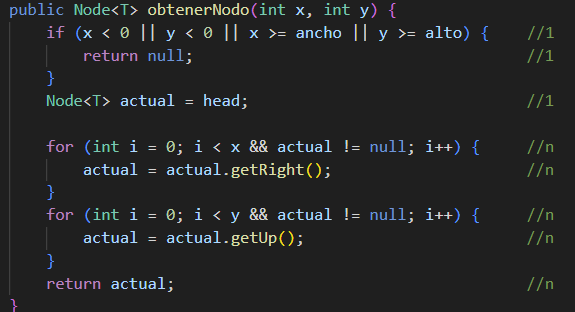
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n^2) ya que recorre un for dentro de otro for dependiendo de las dimensiones de alto como de ancho que tiene la matriz y en cada posición va interconectando los nodos con sus vecinos.

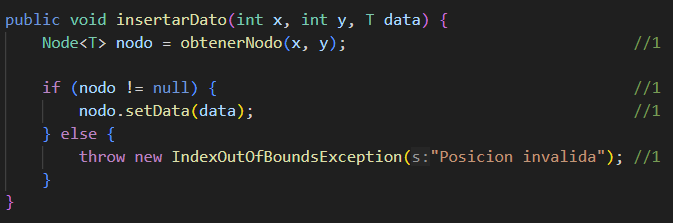
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n) ya que recorre un for n cantidad de veces hasta llegar a una poscion en x y es ahí cuando recorre otro for n cantidad de veces hasta llegar a una posición en y y retorna el dato en la posición.

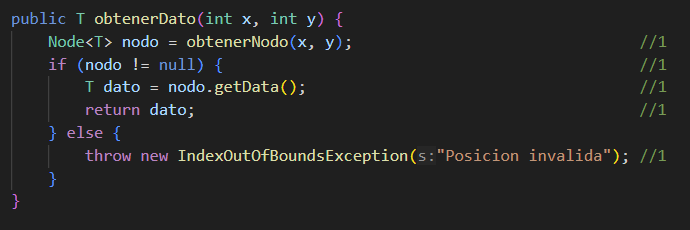
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(1) ya que recibe un dato y lo inserta en la posición recibida como parámetro.

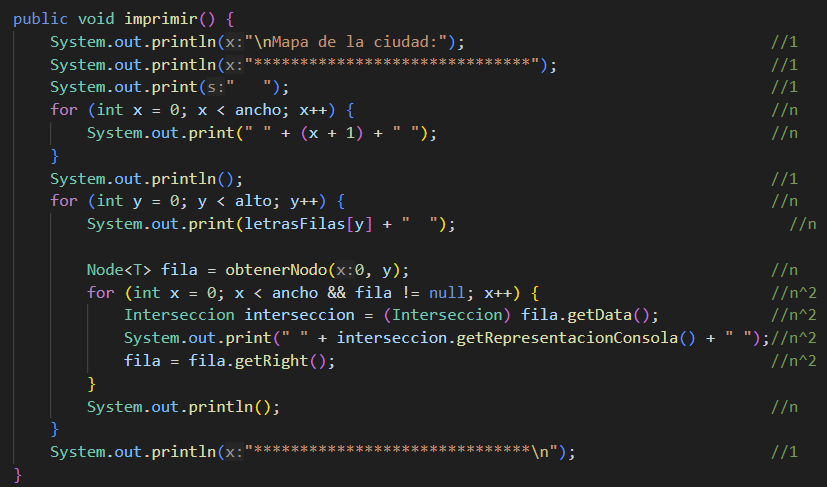
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(1) ya que encuentra un dato en la posición recibida como parámetro y lo devuelve.

* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n^2) ya que recorre un for dentro de otro for dependiendo de las dimensiones de alto como de ancho que tiene la matriz y en cada posición va imprimiendo en consola el dato que haya en la posición.

Clase PriorityQueueI

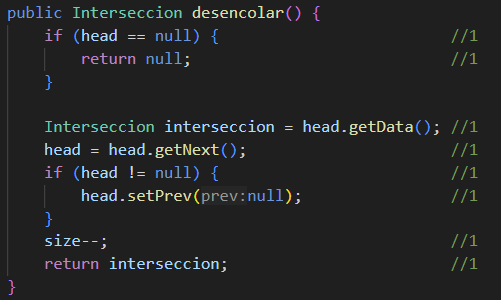
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n) ya que inserta un dato en la posicion n hasta que se encuentra la posición adecuada para colocarse, dada la complejidad de la intersección.

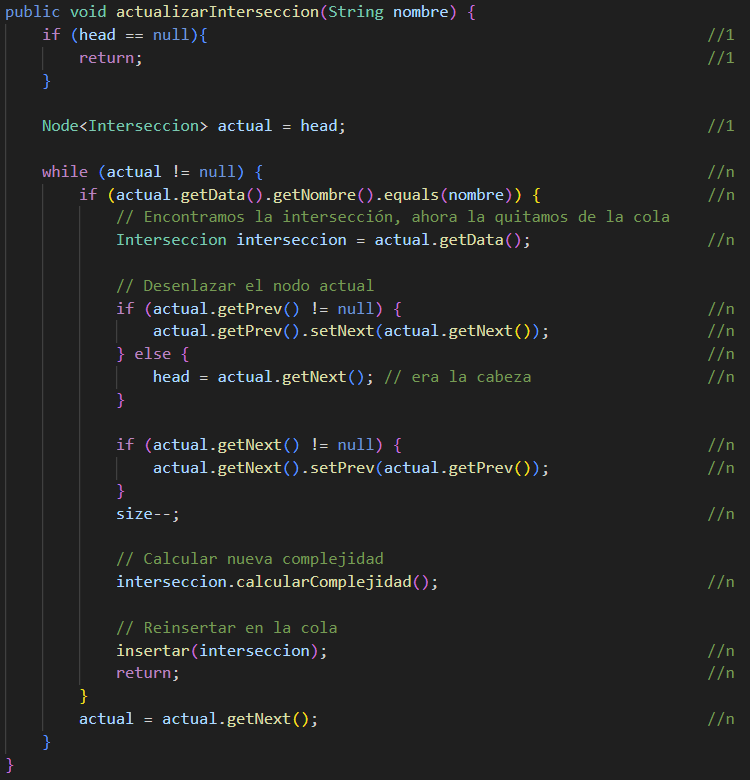
* Método:



Justificación de complejidad:

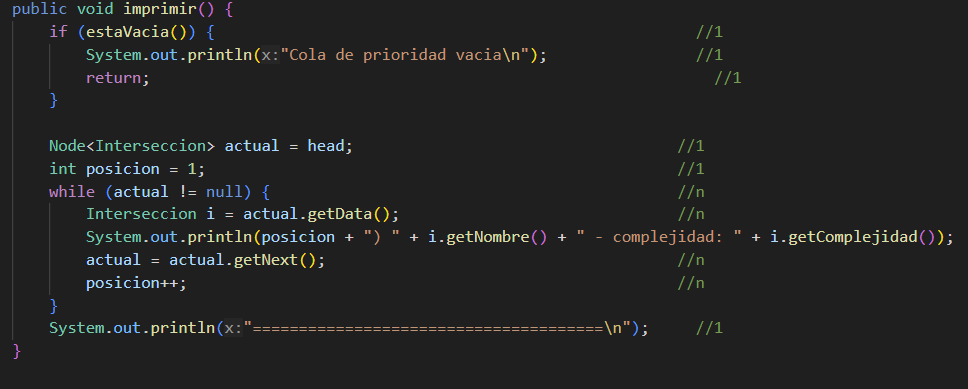
Este método tiene una complejidad de O (1) ya que toma el primer elemento de la cola y lo devuelve.

* Método:



Justificación de complejidad: Este metodo tiene una complejidad de O(n) ya que recorre una cola en búsqueda de una intersección usando el nombre como id, para desencolarlo, calcular su complejidad y volver a encolarlo.

* Método:

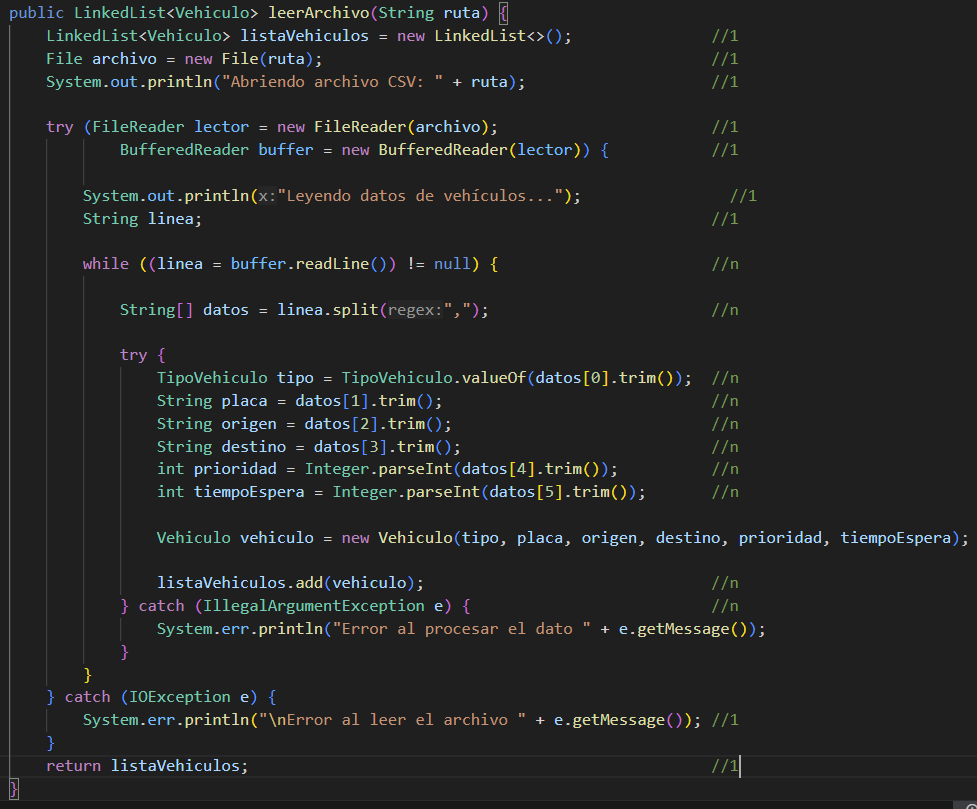


Justificación de complejidad:

Este metodo tiene una complejidad de O(n), ya que recorre una lista n veces y en cada posición muestra en consola el dato del nodo en el que este iterando.

**Clase Archivo**

* Método:

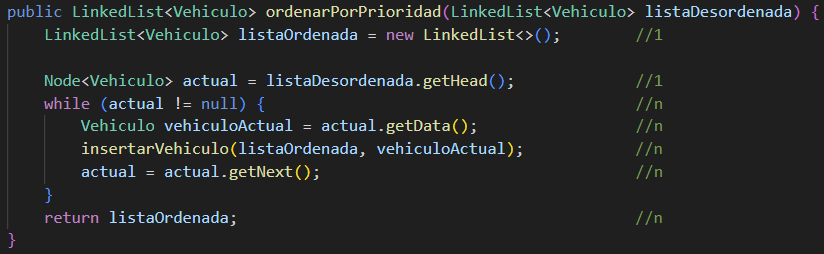


Justificación de complejidad:

Este metodo tiene la complejidad O(n) ya que recorre un while n veces dependiendo de la cantidad de vehículos haya en el archivo y por cada vehiculo o línea crea la instancia y guarda el vehiculo en la lista.

**Clase InserscionDirecta**

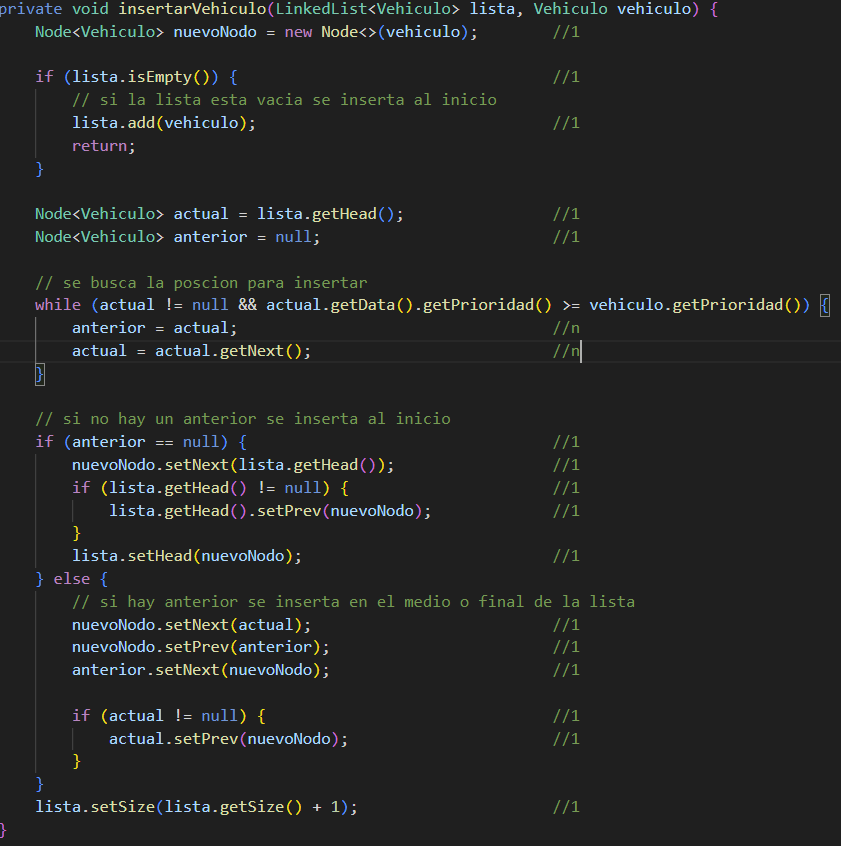
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n^2) ya que recorre una lista n veces e inserta cada elemento de manera ordenada para asi retornar otra lista con los datos ya ordenados, pero en el proceso de ordenación recorre otra lista para buscar la posición correcta.

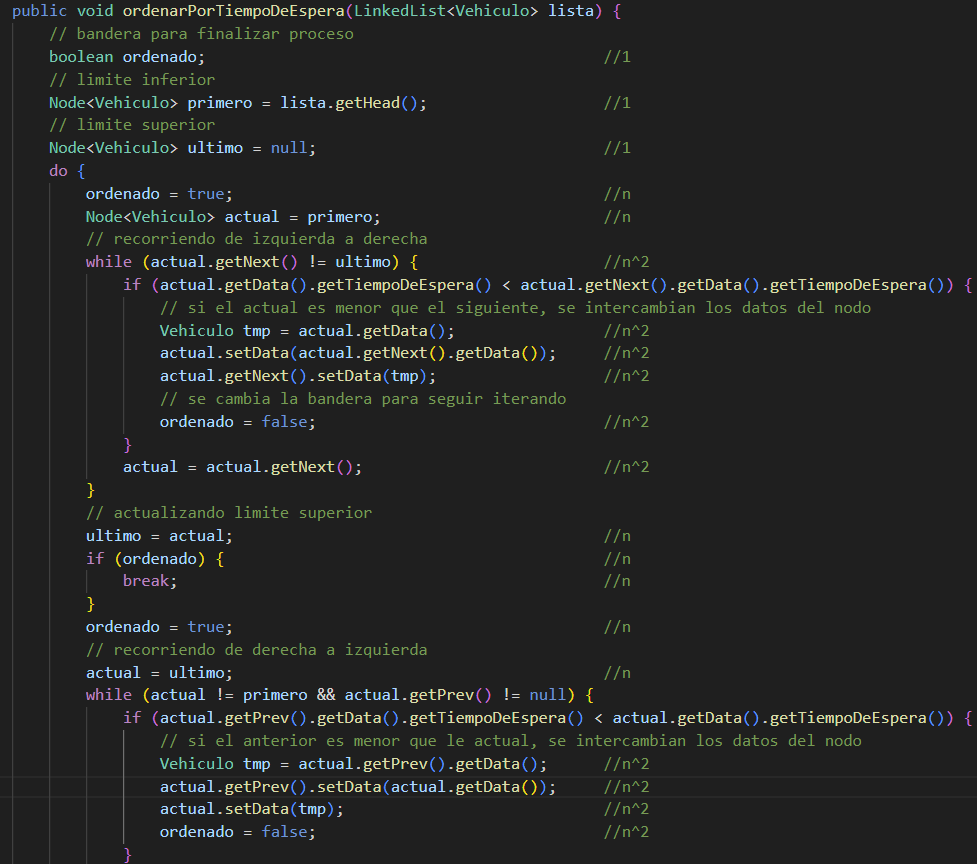
* Método:



Justificación de complejidad:  
este metodo tiene la complejidad de O(n) ya que inserta un vehiculo en una lista, pero recorre la lista n veces en busca de su mejor posición, y lo coloca ahí.

**Clase MetodoDeLaSacudida**

* Método:

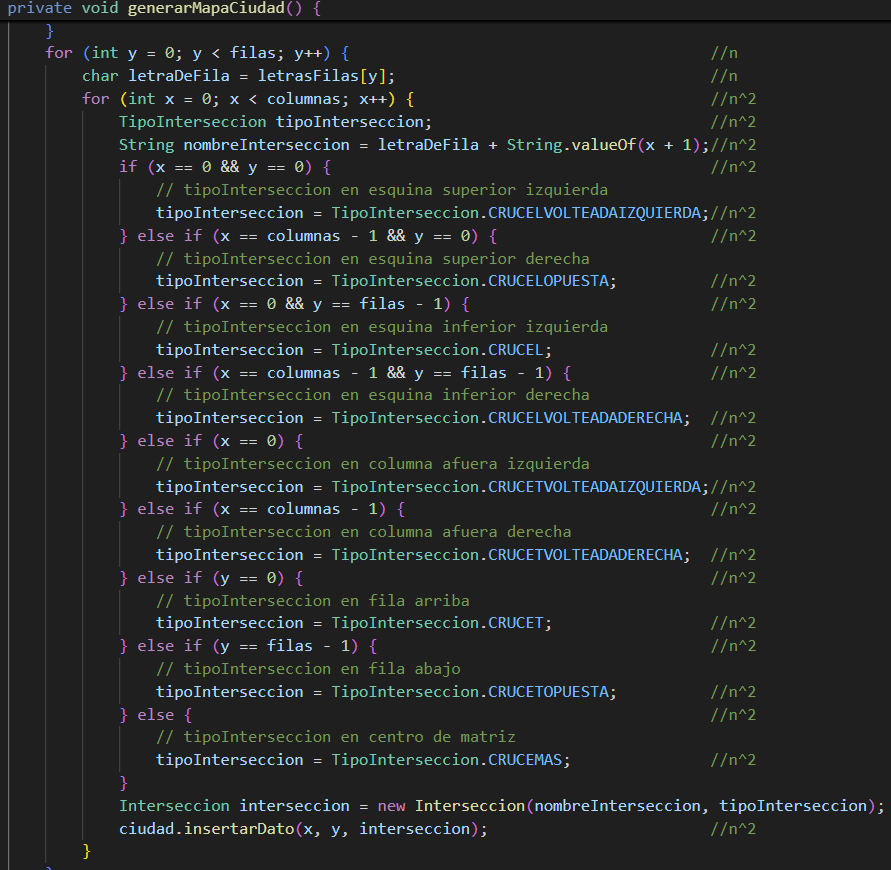


Justificación de complejidad:

Este metodo tiene la complejidad de O(n^2) ya que itera con un while las veces que sea necesaria, pero en cada iteración hace un recorrido de izquierda a derecha y si es necesario de derecha a izquierda n veces hasta que la lista se encuentre totalmente ordenada.

Clase Simulador

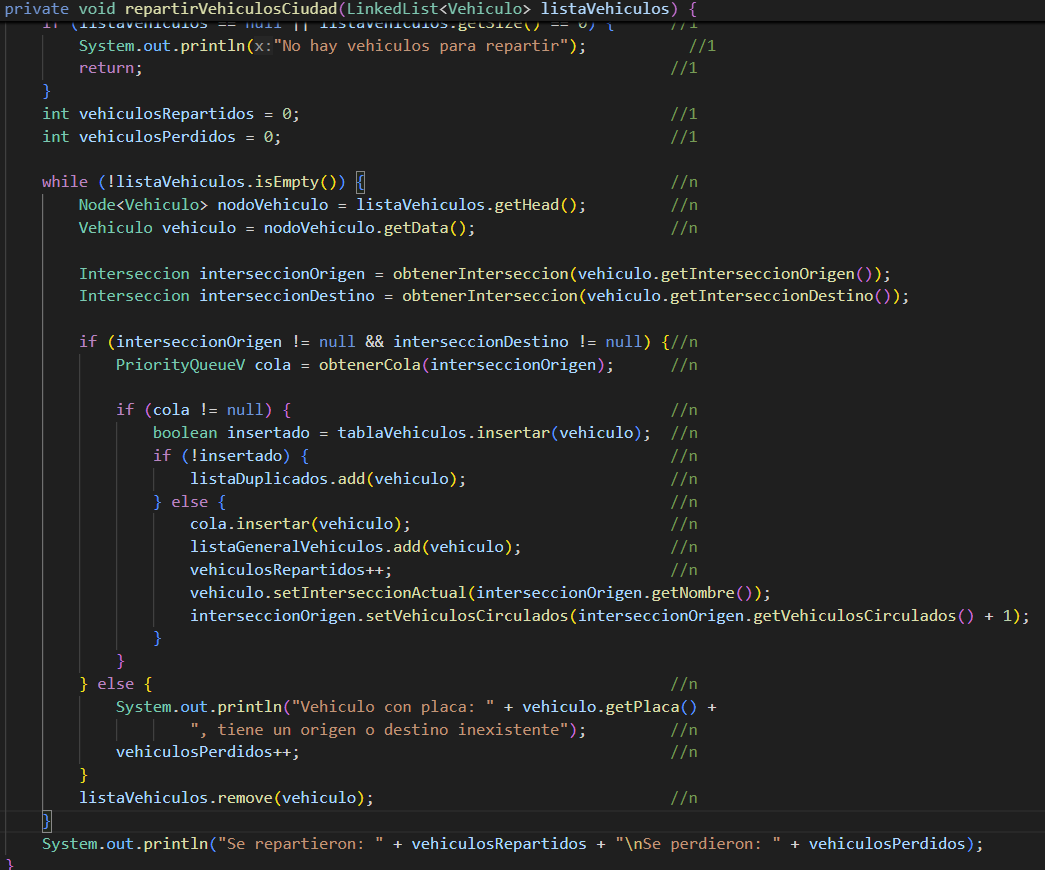
* Método:



Justificación de complejidad:

Este metodo tiene una complejidad de O(n^2) ya que recorre un for dentro de otro for dependiendo de las dimensiones de alto como de ancho que tiene la matriz y en cada posición va insertando intersecciones según las posiciones en que se este iterando.

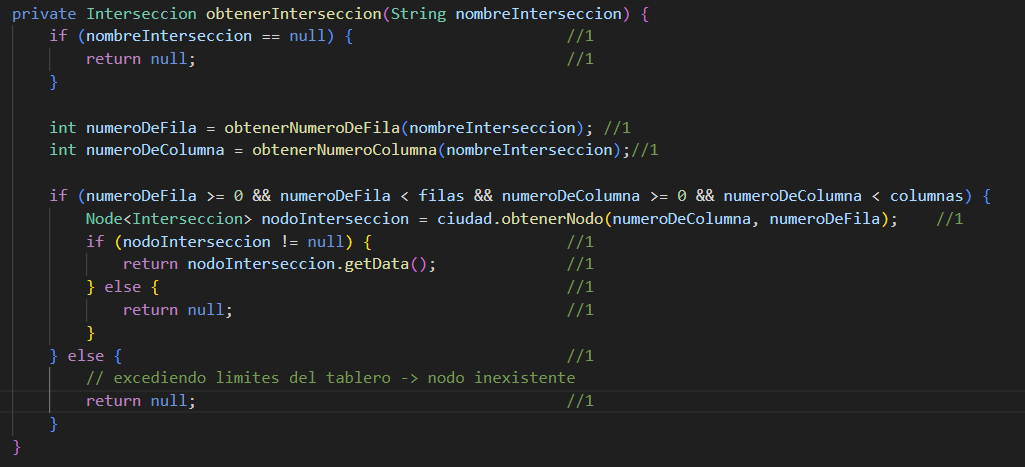
* Método:



Justificación de complejidad:

Este metodo tiene la complejidad de O (n) ya que recorre una lista de vehículos n veces hasta vaciarla para que con cada vehiculo lo inserte en la intersección que tiene como origen.

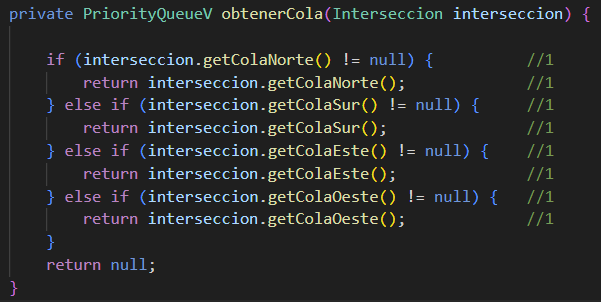
* Método:



Justificación de complejidad:

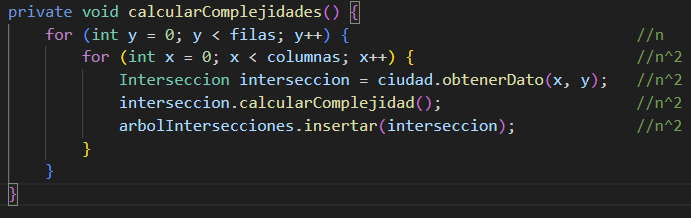
Este metodo tiene la complejidad de O(1) ya que dadas unas coordenadas retorna la intersección que se encuentre en esa posición de las coordenadas.

* Método:



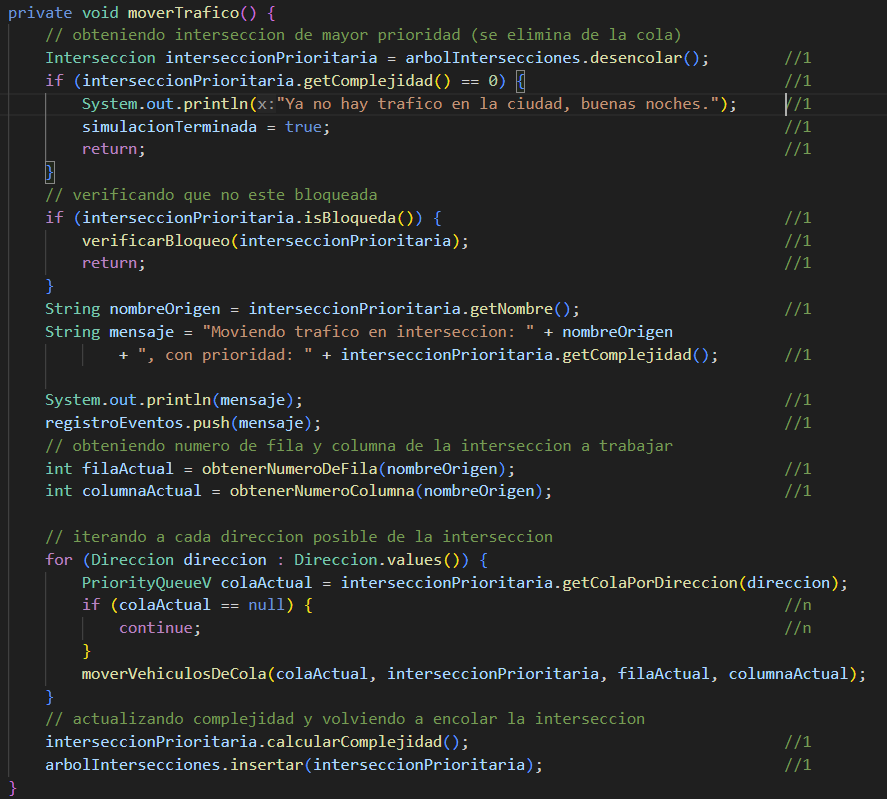
Justificación de complejidad:

* Método:



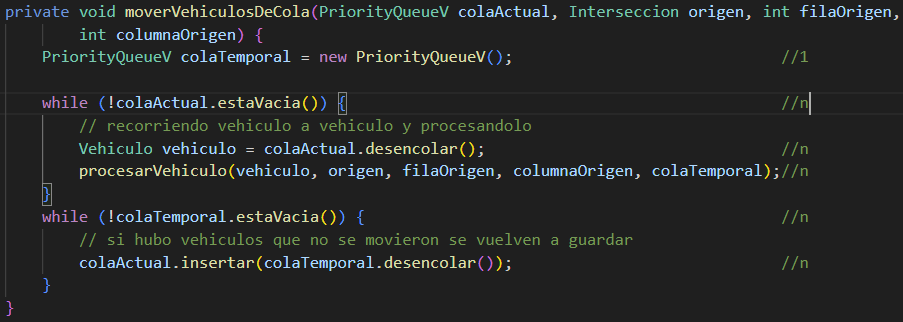
Justificación de complejidad:

* Método:



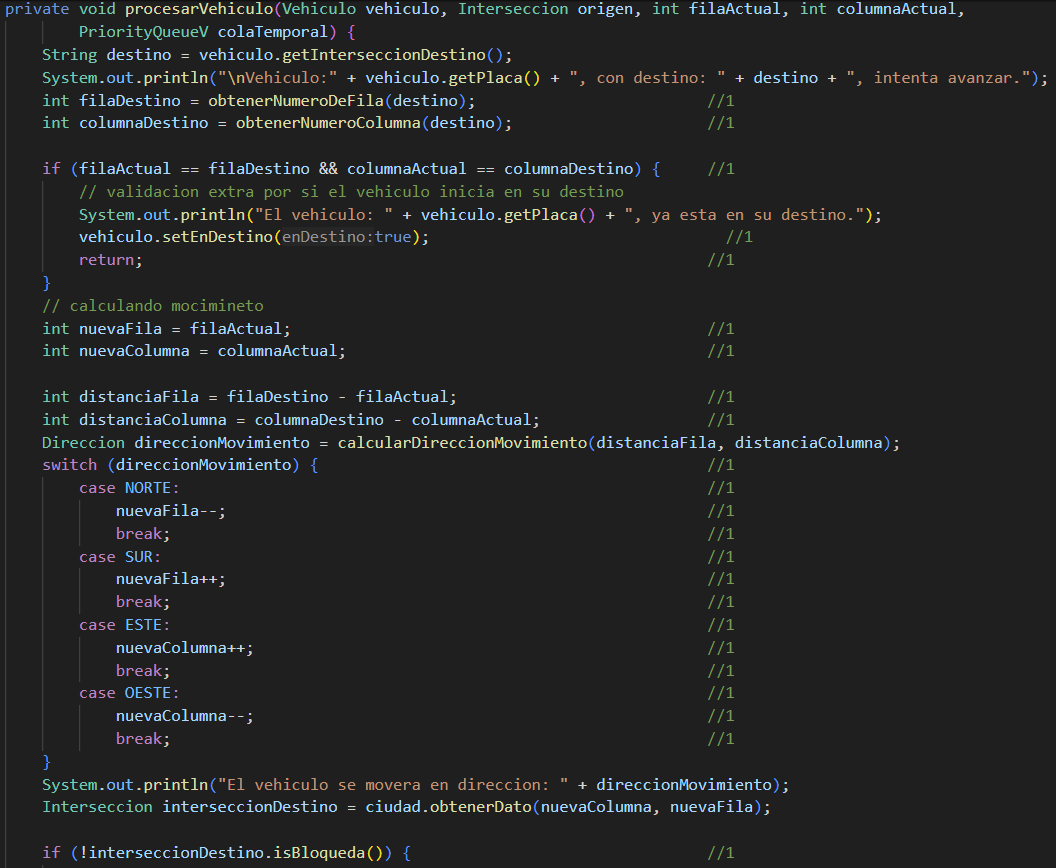
Justificación de complejidad:

* Método:



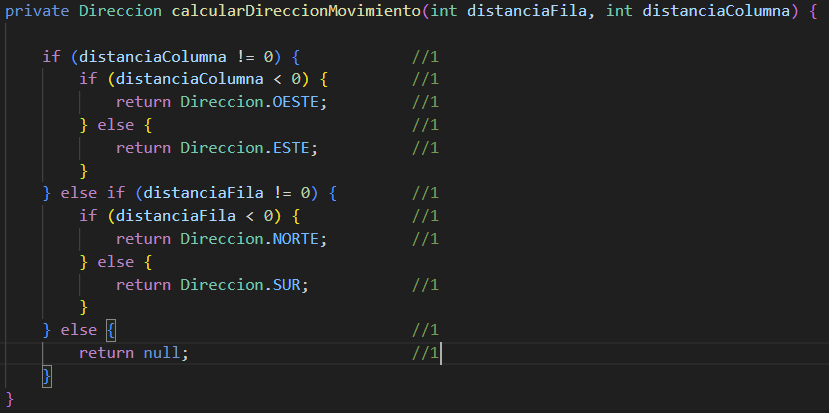
Justificación de complejidad:

* Método:



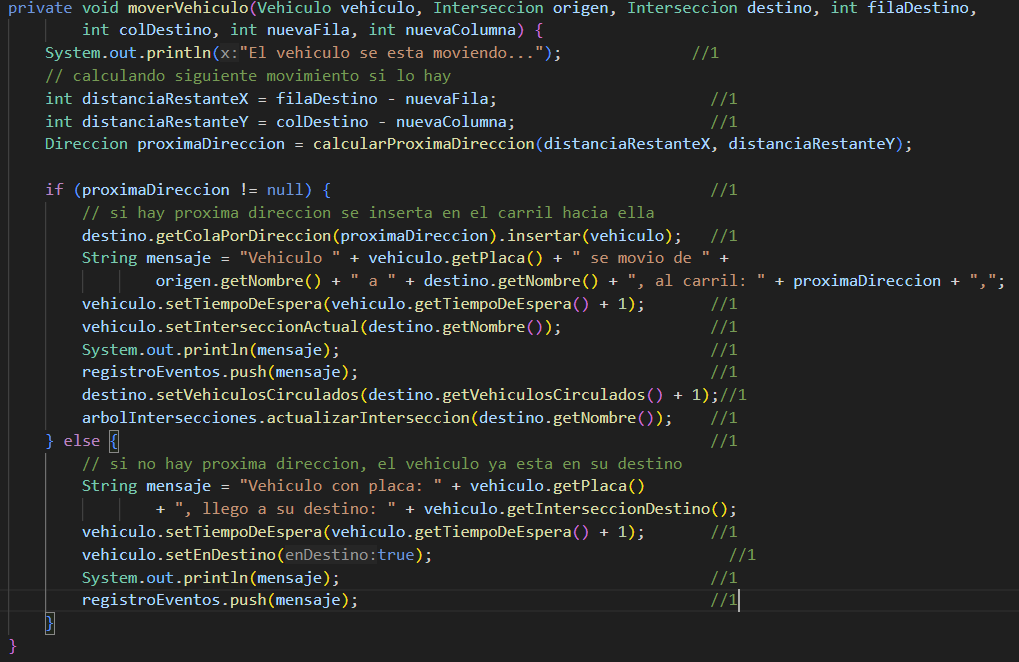
Justificación de complejidad:

* Método:



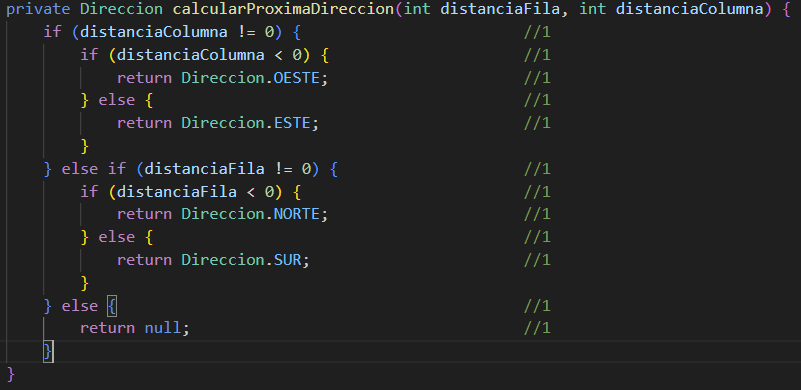
Justificación de complejidad:

* Método:



Justificación de complejidad:

* Método:



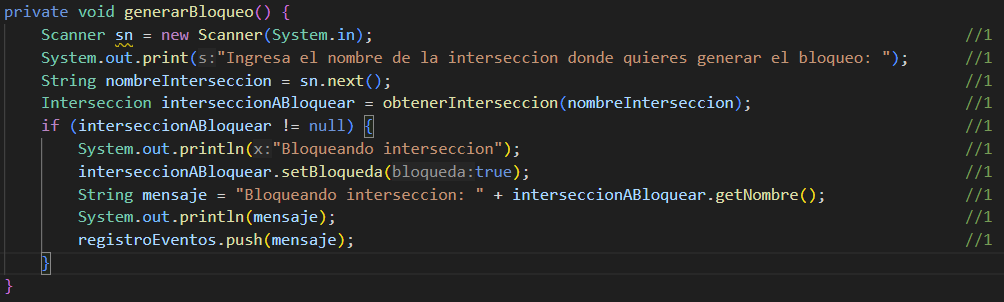
Justificación de complejidad:

* Método:



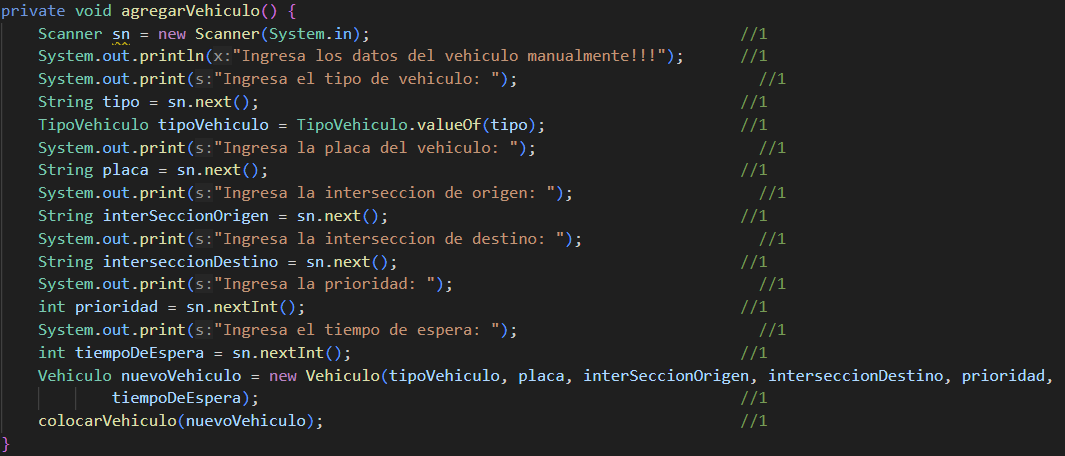
Justificación de complejidad:

* Método:



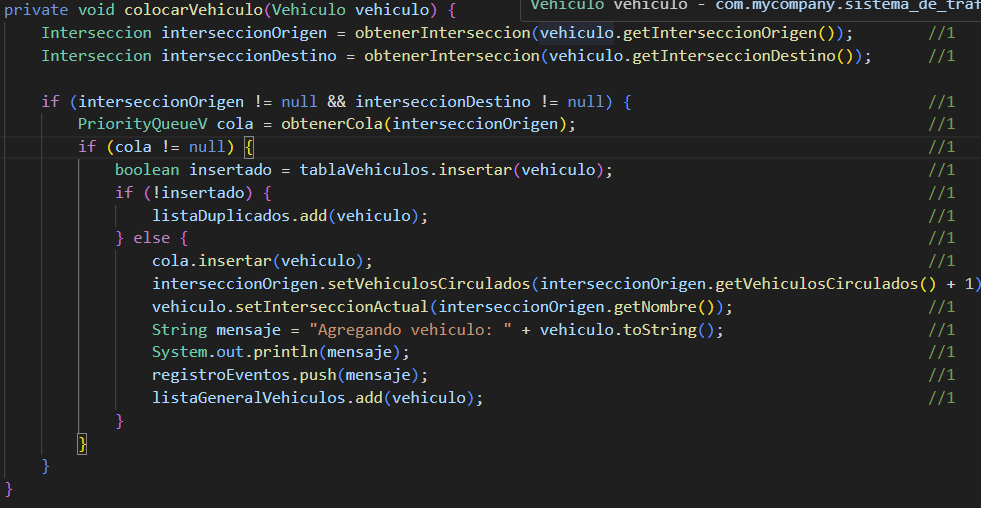
Justificación de complejidad:

* Método:



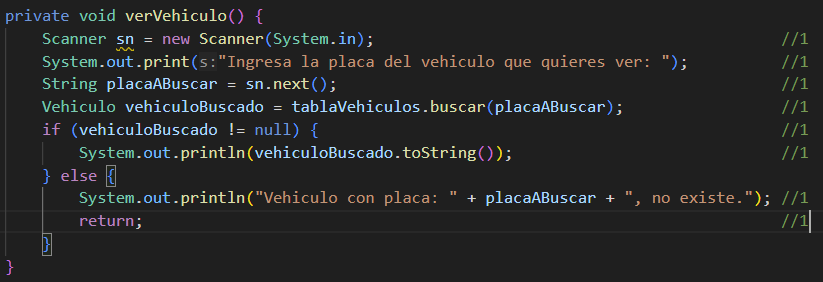
Justificación de complejidad:

* Método:



Justificación de complejidad:

* Método:



Justificación de complejidad: