|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Ing. Guadalupe Lizeth Parrales Romay |
| *Asignatura:* | Programación Orientada a Objetos |
| *Grupo:* | 04 |
| *No de Práctica(s):* | 13 |
| *Integrante(s):* | Dávila Ortega Jesús Eduardo - No. Cuenta: 317199860  Díaz Hernández Marcos Bryan - No. Cuenta: 317027253  Pareja Ávila Emiliano - No. Cuenta: 317081345  Vázquez Zavala Oliver Alexis - No. Cuenta: 317202263 |
| *No. de Equipo de cómputo empleado:* |  |
| *No. de Lista o Brigada:* | 12,14, 30, 37 |
| *Semestre:* | 2021-1 |
| *Fecha de entrega:* | 26/01/2021 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_

**Objetivo de la práctica.**

Implementar una aplicación en un lenguaje orientado a objetos utilizando algún patrón de diseño.

**Introducción.**

Podemos entender un patrón como una descripción del problema y la esencia de su solución de tal forma que ésta se pueda reutilizar en diferentes casos, es, por lo tanto, una solución dada a un problema común. En la ingeniería de software un patrón de diseño resulta ser una solución a un problema de diseño, para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores, otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Los patrones de diseño pretenden proporcionar catálogos de elementos reusables en el diseño de sistemas de software, además de evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente para así normalizar un vocabulario común entre diseñadores y estandarizar el modo en que se realiza el diseño, permitiendo facilitar de cierta manera el diseño de software ayudando a prevenir errores que pueden afectar a todo un proyecto.

Podemos clasificar los patrones en base a tres niveles o propósitos:

* Patrones de arquitectura: Aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.
* Patrones de diseño: Aquellos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.
* Patrones de programación: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

Además, los patrones de diseño se pueden clasificar en tres grandes grupos que son patrones creacionales, patrones estructurales y patrones de comportamiento. Los elementos de un patrón de diseño son:

-Nombre: describe el problema de diseño.

-Contexto o motivación: Escenario de ejemplo para la aplicación del patrón.

-Aplicabilidad o Consideraciones: Usos comunes y criterios de aplicabilidad del patrón.

-El problema: describe el problema sobre el cual aplicar el patrón.

-La solución: describe los elementos que componen el diseño, sus relaciones, responsabilidades y colaboración.

-Consecuencias: Consecuencias positivas y negativas en el diseño derivadas de la aplicación del patrón.

Un ejemplo de lo anterior en el paradigma orientado a objetos es el llamado patrón de diseño MVC, el cual parte de las iniciales de Modelo-Vista-Controlador (Model-View-Controller, en inglés), este patrón propone, independientemente de las tecnologías o entornos en los que se base el sistema a desarrollar, la separación de los componentes de una aplicación en tres grupos (o capas) principales: el modelo, la vista, y el controlador, y describe cómo se relacionarán entre ellos para mantener una estructura organizada, limpia y con un acoplamiento mínimo entre las distintas capas.

En la capa Modelo encontraremos siempre una representación de los datos del dominio, es decir, aquellas entidades que nos servirán para almacenar información del sistema que estamos desarrollando. Por ejemplo, si estamos desarrollando una aplicación de facturación, en el modelo existirán las clases Factura, Cliente o Proveedor, entre otras.

Los componentes de la Vista son los responsables de generar la interfaz de nuestra aplicación, es decir, de componer las pantallas, páginas, o cualquier tipo de resultado utilizable por el usuario o cliente del sistema. De hecho, suele decirse que la Vista es una representación del estado del Modelo en un momento concreto y en el contexto de una acción determinada.

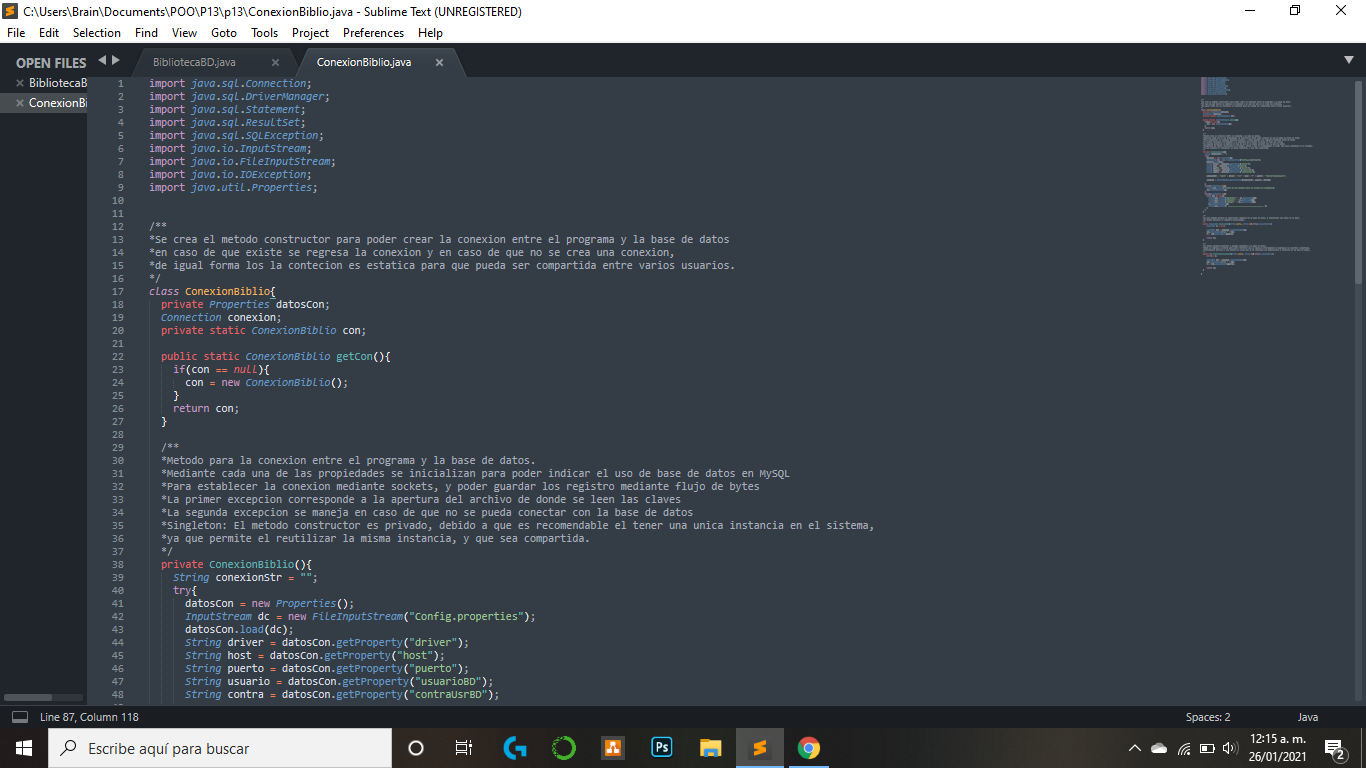
Y por último tenemos al controlador cuya tarea es actuar como intermediarios entre el usuario y el sistema, sus componentes serán capaces de capturar las acciones de éste sobre la vista, como puede ser, por ejemplo, la pulsación de un botón o la selección de una opción de menú, interpretarlas y actuar en función de ellas, por ejemplo, retornando al usuario una nueva vista que represente el estado actual del sistema, o invocando a acciones definidas en el modelo para consultar o actualizar información, de igual manera existen más patrones de diseño que pueden implementarse al desarrollar una determinada aplicación, según sea el caso, por lo cual es de gran ayuda el conocer sobre los patrones de diseño para así agilizar el proceso de desarrollo de solución.

**Análisis de los ejercicios.**

**Ejercicio 1: Biblioteca**

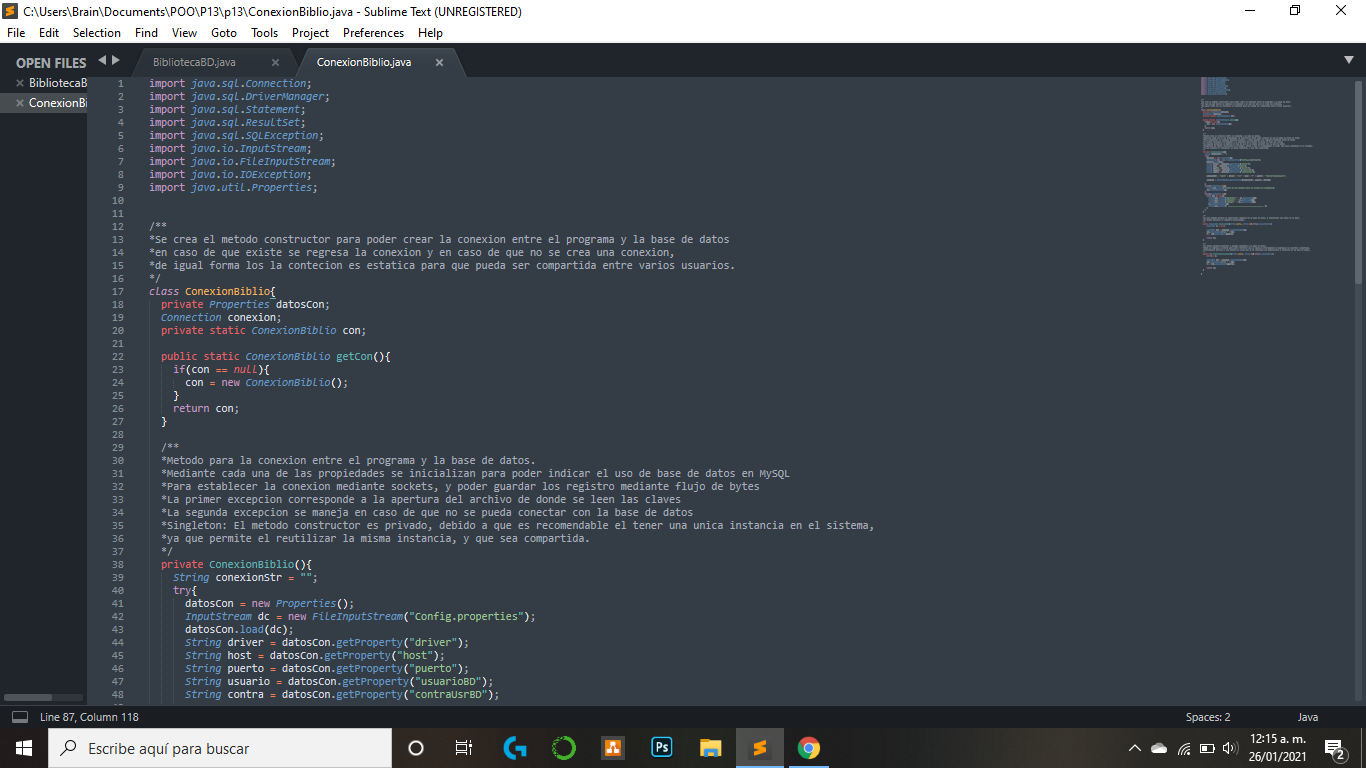
* **Análisis**

La clase ConexionBiblioteca permite la conexión con una base de datos mediante sql, cada uno de los métodos permite establecer la comunicación entre la base de datos y el programa.



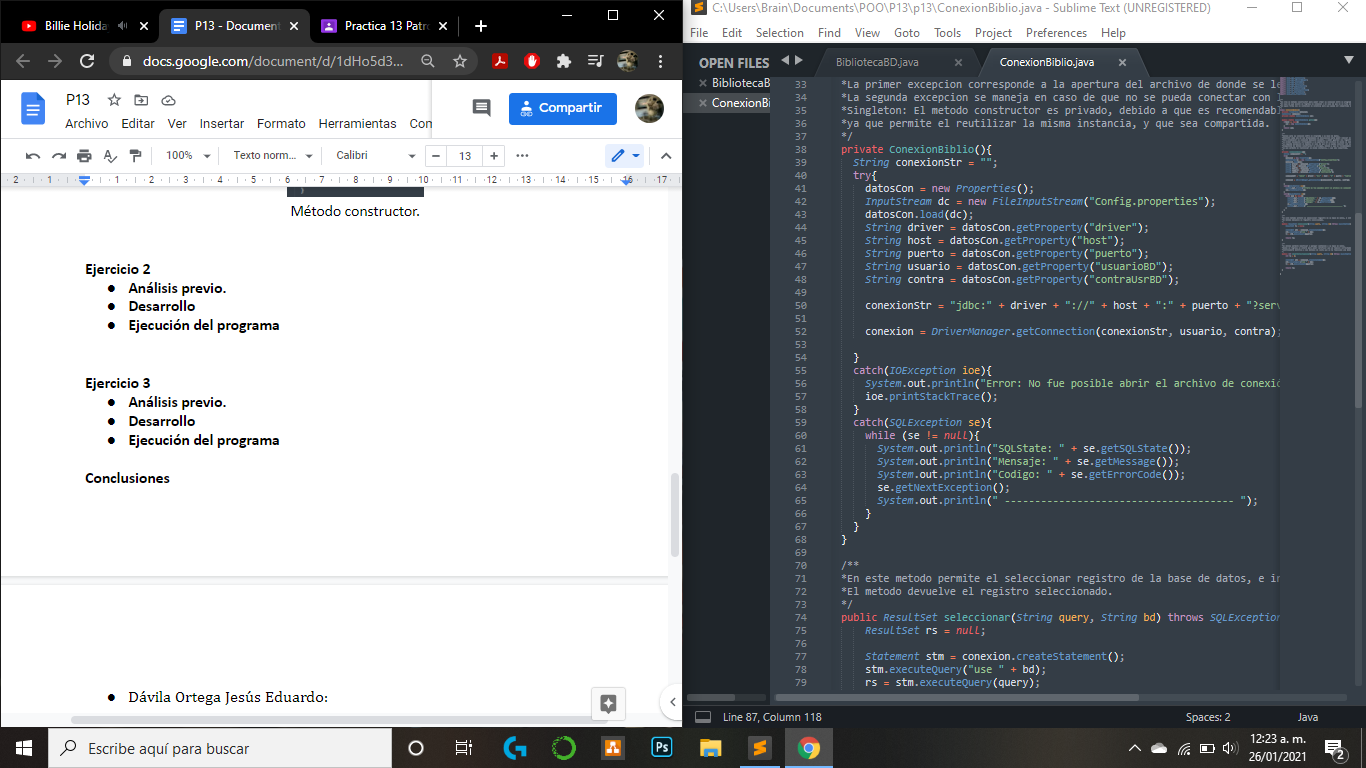
Importación de las clases.

El método ConexionBiblio getCon(), regresa la conexión creada, de esta forma se puede compartir una única instancia y no se tienen varias instancias que modifiquen la misma información. En caso de que no exista una conexión se crea una y se regresa.



Método ConexionBiblio getCon().

El método constructor permite la creación de una conexión con la base de datos, una vez esté establecida la conexión esta permite modificar los registros que se encuentran en la base de datos. Al ser un método privado, instancias de otras clases no pueden acceder a él, por esto es que se utiliza una única conexión con la base de datos, lo que permite reutilizar la instancia.



Método constructor.

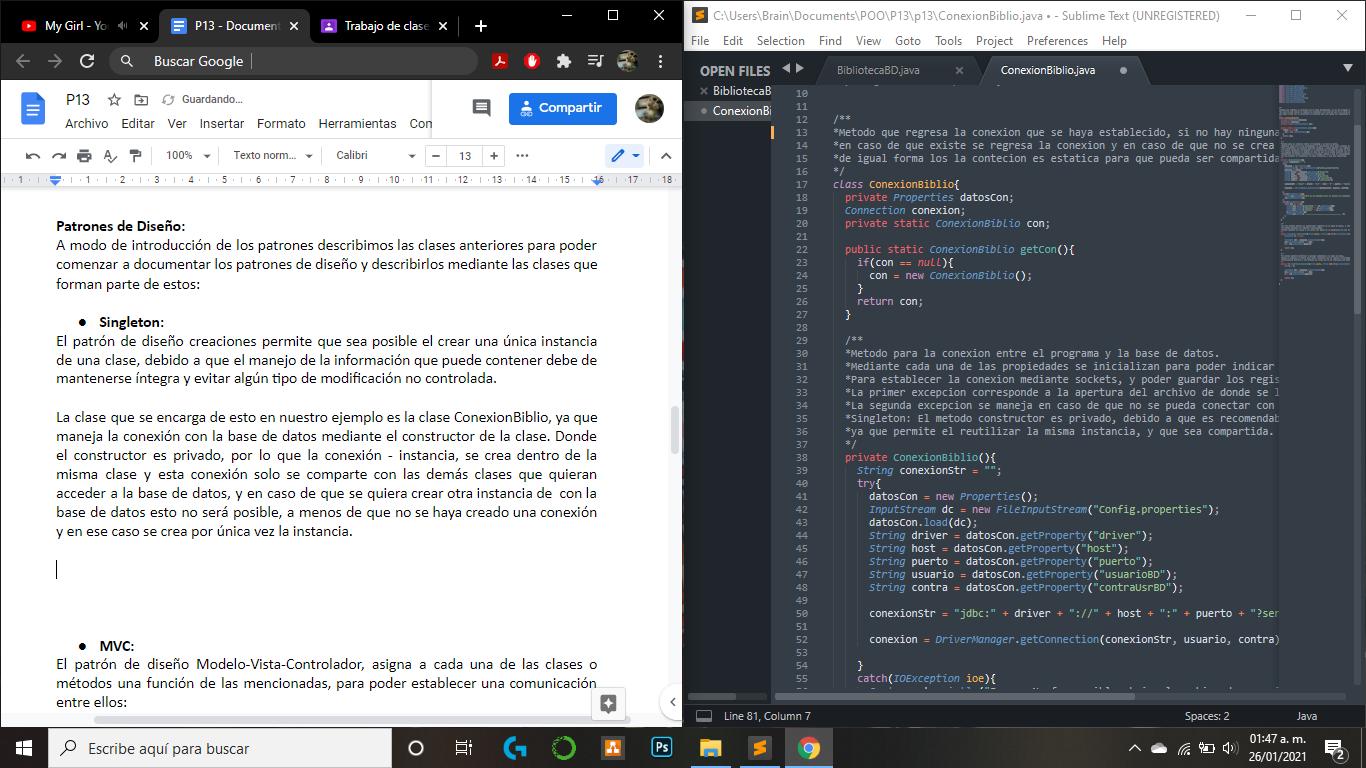
**Patrones de Diseño:**

A modo de introducción de los patrones describimos las clases anteriores para poder comenzar a documentar los patrones de diseño y describirlos mediante las clases que forman parte de estos:

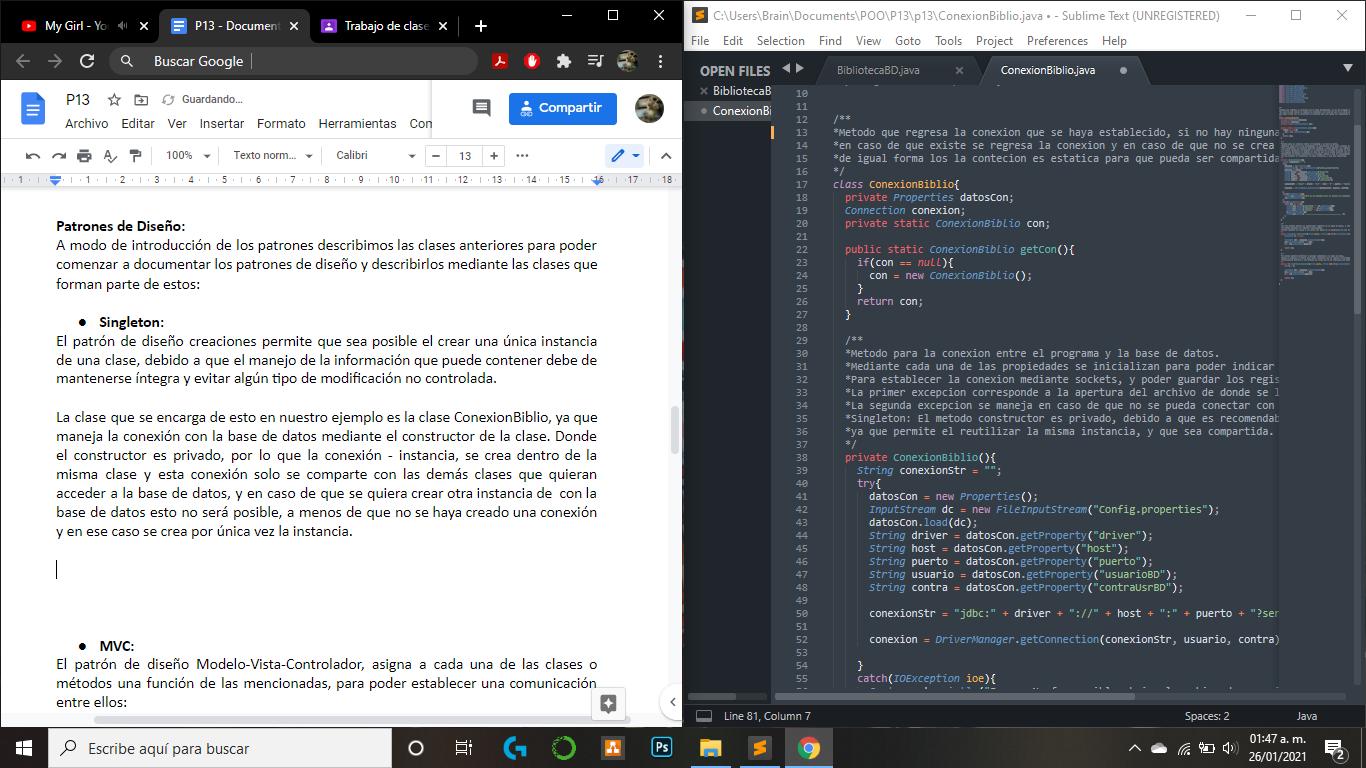
* **Singleton:**

El patrón de diseño creaciones permite que sea posible el crear una única instancia de una clase, debido a que el manejo de la información que puede contener debe de mantenerse íntegra y evitar algún tipo de modificación no controlada.

La clase que se encarga de esto en nuestro ejemplo es la clase ConexionBiblio, ya que maneja la conexión con la base de datos mediante el constructor de la clase. Donde el constructor es privado, por lo que la conexión - instancia, se crea dentro de la misma clase y esta conexión solo se comparte con las demás clases que quieran acceder a la base de datos, y en caso de que se quiera crear otra instancia de con la base de datos esto no será posible, a menos de que no se haya creado una conexión y en ese caso se crea por única vez la instancia.



Método getCon().



Método Constructor().

Por este patrón de diseño la instancia que se crea: ConexionBiblio con; es estática lo que le permite que sea compartida y mediante el método getCon() se puede enviar a donde se necesite la conexión con la base de datos.

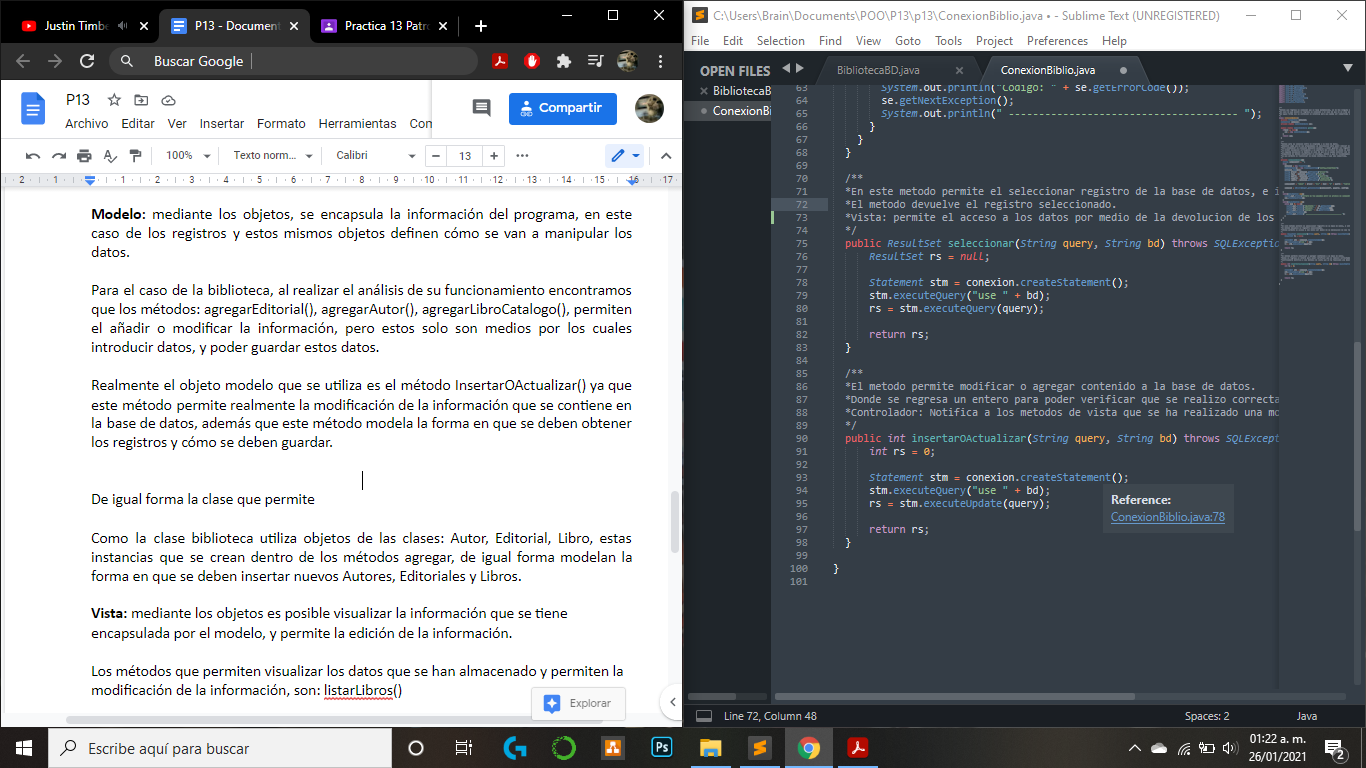
* **MVC:**

El patrón de diseño Modelo-Vista-Controlador, asigna a cada una de las clases o métodos una función de las mencionadas, para poder establecer una comunicación entre ellos:

**Modelo:** mediante los objetos, se encapsula la información del programa, en este caso de los registros y estos mismos objetos definen cómo se van a manipular los datos.

Para el caso de la biblioteca, al realizar el análisis de su funcionamiento encontramos que los métodos: agregarEditorial(), agregarAutor(), agregarLibroCatalogo(), permiten el añadir o modificar la información, pero estos solo son medios por los cuales introducir datos, y poder guardar estos datos.

Realmente el objeto modelo que se utiliza es el método InsertarOActualizar() ya que este método permite realmente la modificación de la información que se contiene en la base de datos, además que este método modela la forma en que se deben obtener los registros, permitiendo la encapsulación de la información.

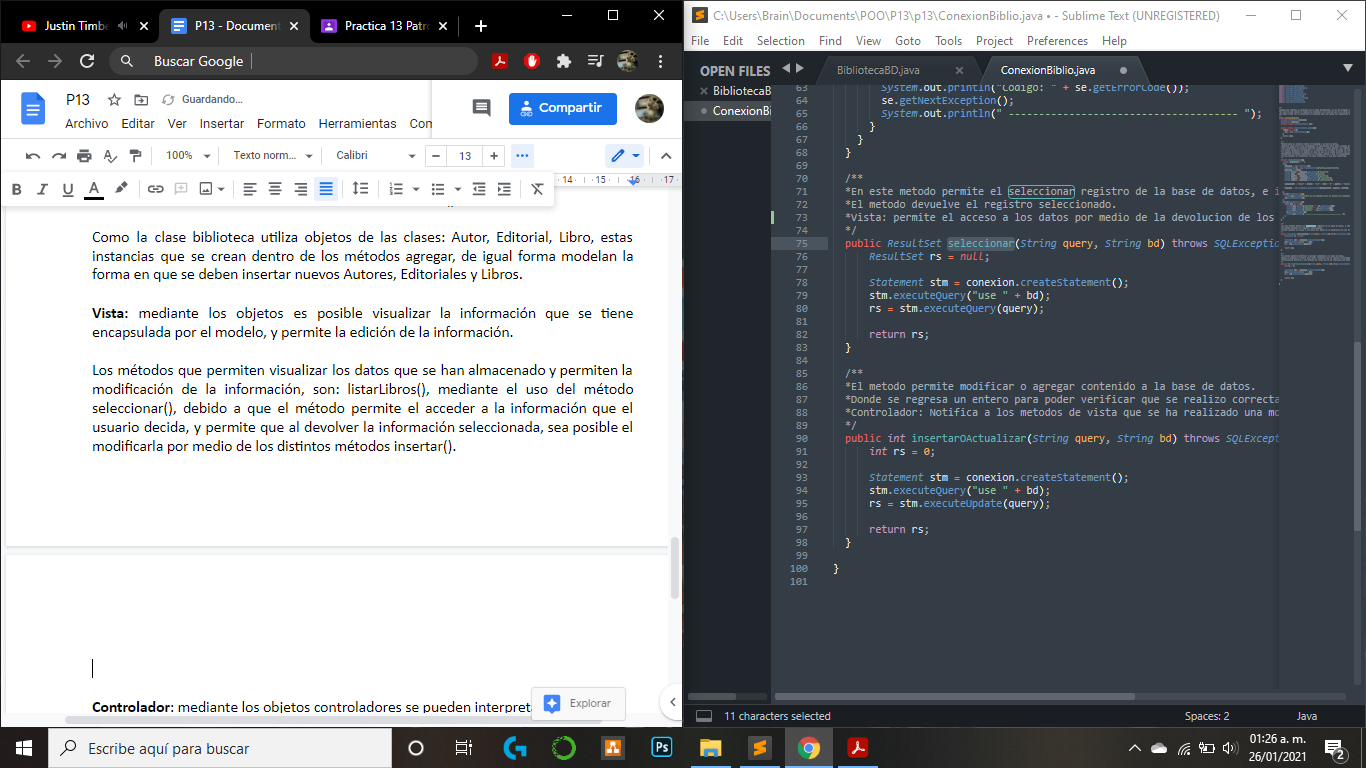


Método InsertarOActualizar()

Como la clase biblioteca utiliza objetos de las clases: Autor, Editorial, Libro, estas instancias que se crean dentro de los métodos agregar, de igual forma modelan la forma en que se deben insertar nuevos Autores, Editoriales y Libros.

**Vista:** mediante los objetos es posible visualizar la información que se tiene encapsulada por el modelo, y permite la edición de la información.

Los métodos que permiten visualizar los datos que se han almacenado y permiten la modificación de la información, son: listarLibros(), mediante el uso del método seleccionar(), debido a que el método permite el acceder a la información que el usuario decida, y permite que al devolver la información seleccionada, sea posible el modificarla por medio de los distintos métodos insertar().

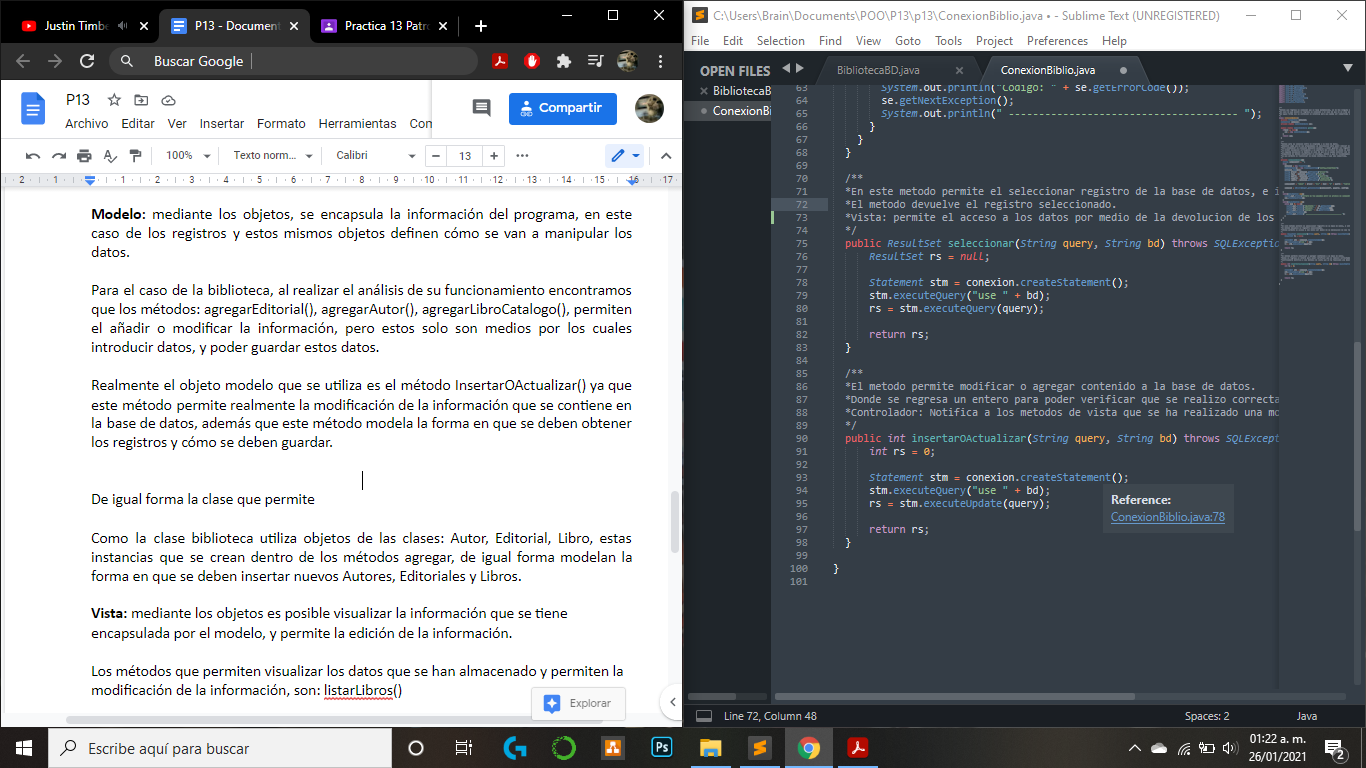


Método Seleccionar()

De igual manera los métodos get() y set(), permiten el obtener la información sobre los libros o los autores, lo que los convierte en tipo vista. Asimismo el método principal permite la visualización de los datos, por ello es también de tipo vista, e incluso permite la ejecución de los métodos que pueden modificar los registros.

**Controlador**: mediante los objetos controladores se pueden interpretar las acciones que realizan los usuarios y se notifica a los objetos de modelo.

Los métodos que corresponden a esta clasificación son los que notifican los cambios a los de tipo modelo, en este caso el mismo método insertarOActualizar() notifica la inserción de un nuevo elemento o la actualización de los registros mediante el regreso de un entero. Posiblemente la conexión con la base de datos es un tipo de controlador ya que con el manejo de las excepciones es posible notificar si los registros eran los adecuados o si existía una base de datos.

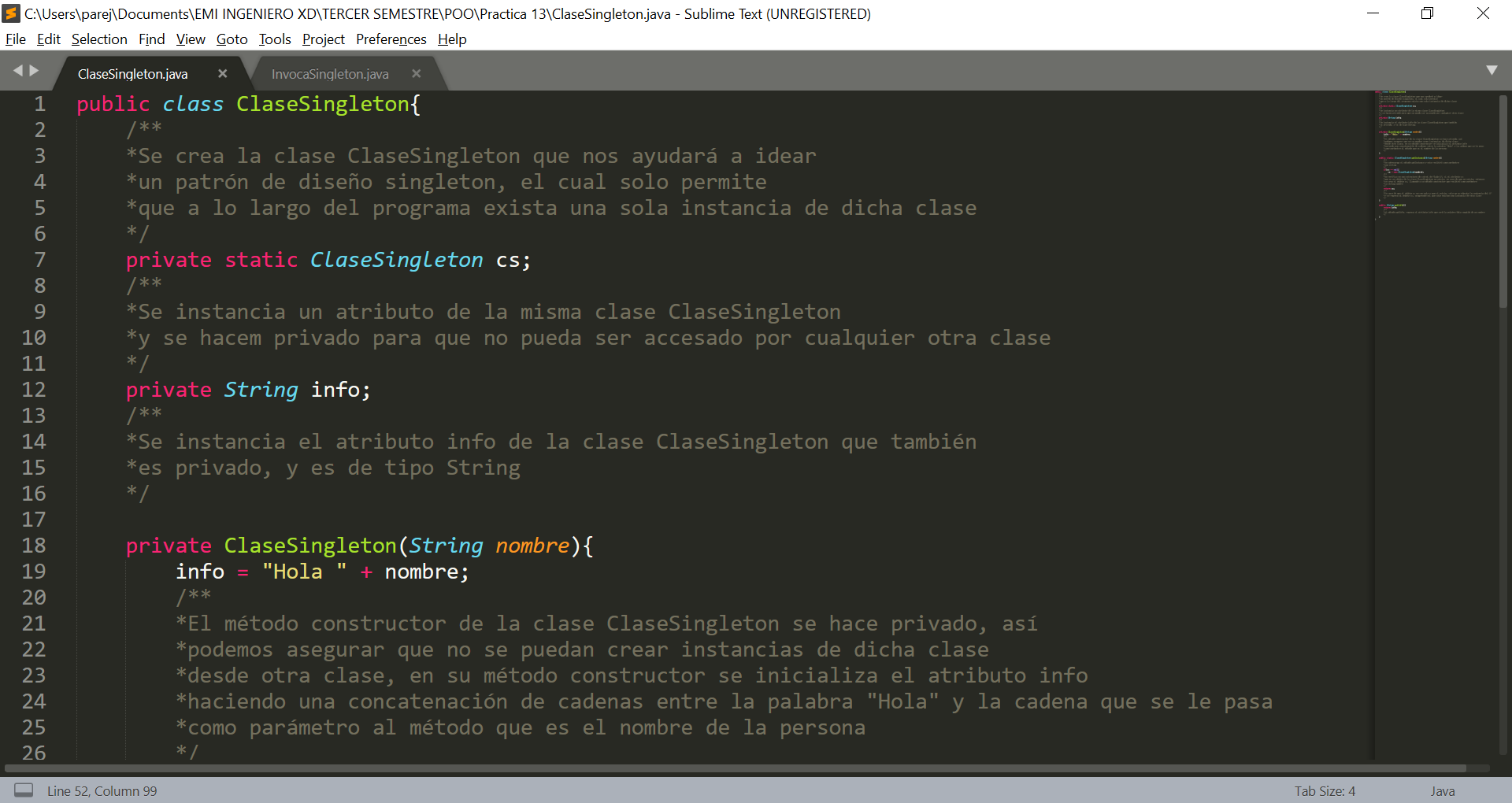


Método InsertarOActualizar()

**Ejercicio 2**

* **Análisis previo.**

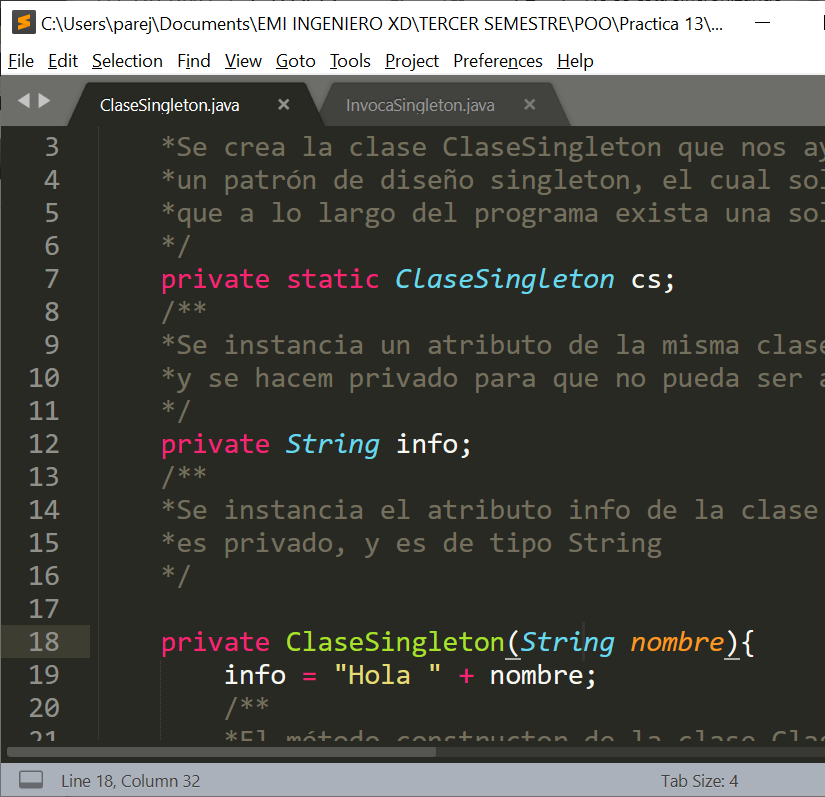
Como se mencionó antes, el patrón de diseño Singleton, es un patrón de diseño creacional, en este diseño, solo debe existir una única instancia de la clase, por tanto esta será general para todas las demás clases que quieran acceder a sus atributos o métodos y es por eso que su método constructor se declara privado, para que solo en la misma clase pueda ser invocado, y no se pueda acceder a él, desde ninguna otra clase. En este caso se observa que la clase ClaseSingleton es la clase que tiene estas propiedades.



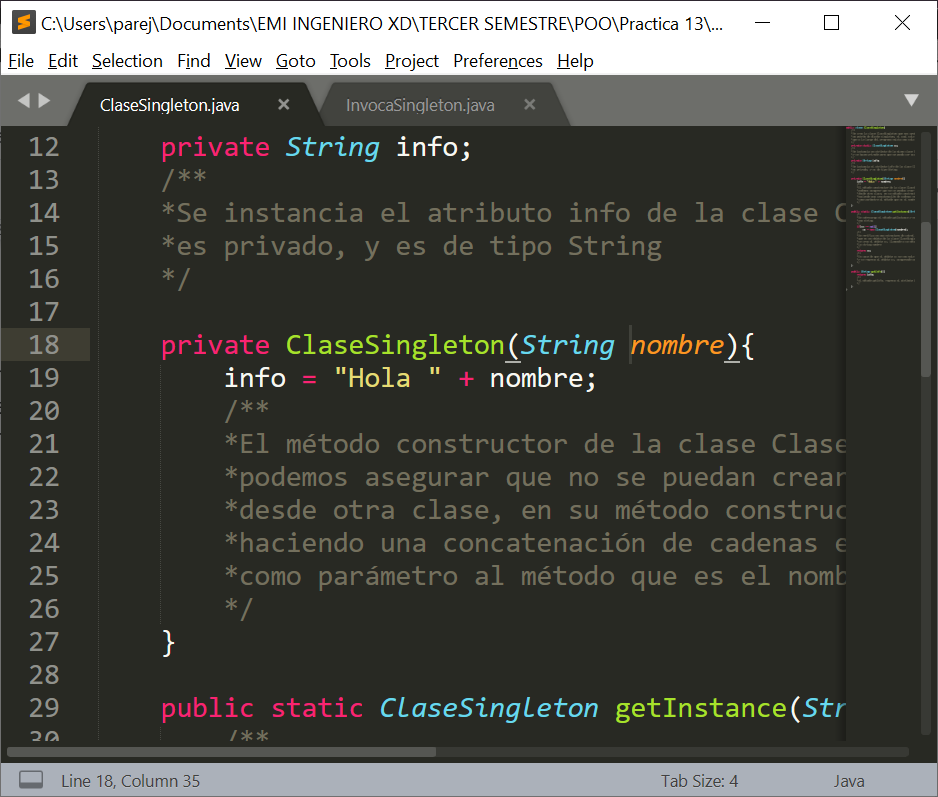
* **Desarrollo**

**Clase ClaseSingleton**

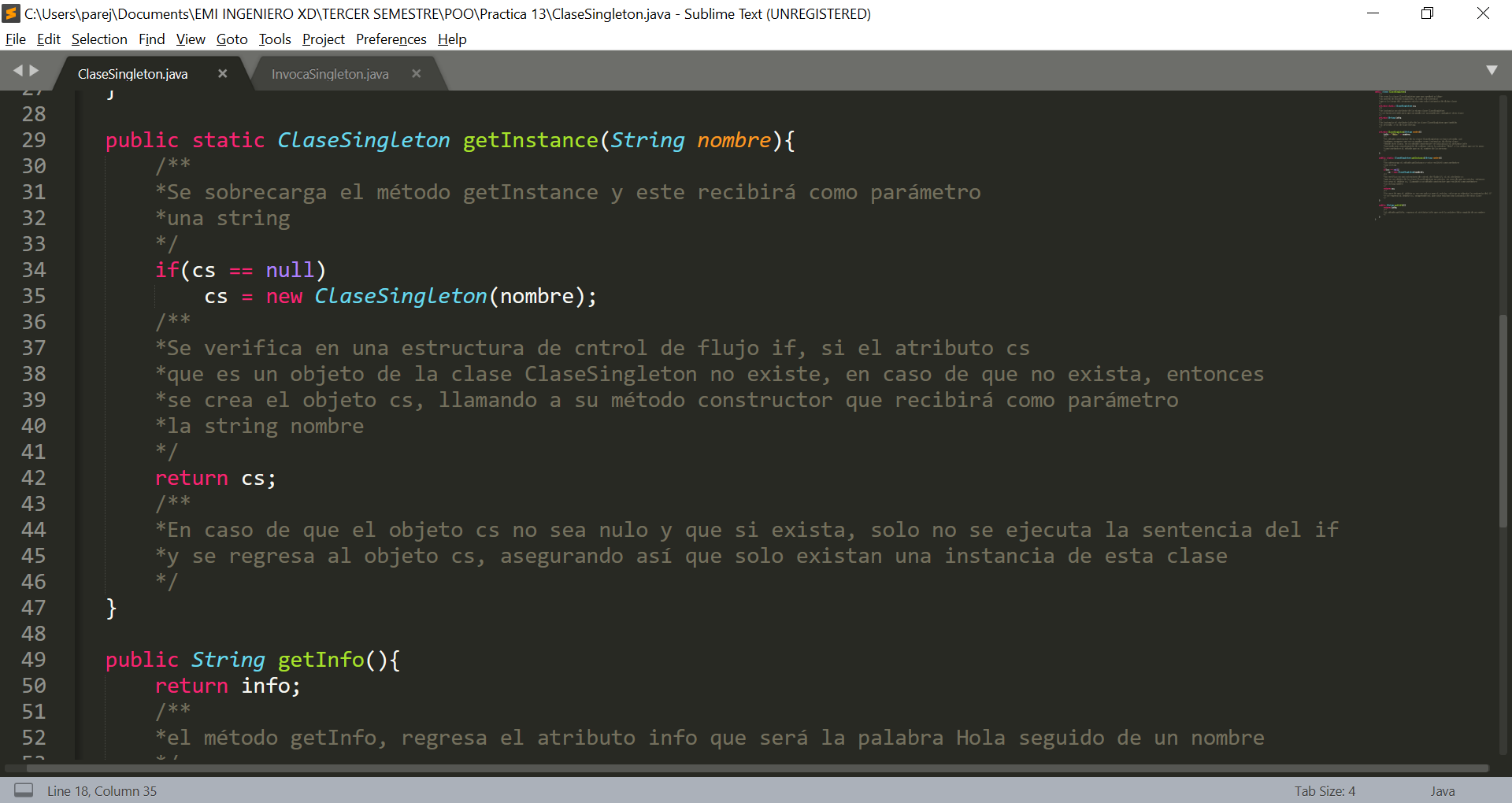
Una vez que se verifica que su método constructor es privado, se observan los atributos de la clase, dentro de esta se crea una instancia de la propia clase ClaseSingleton, que será a partir de la cual generaremos la única instancia de la clase durante todo el programa, después hay un atributo de tipo string llamado info.



Su método constructor, que es privado para evitar que se creen instancias de esta clase en otras clases que no sea la propia, recibe como parámetro una cadena llamada nombre, y dentro de este método, solo se inicializa el atributo info concatenando dos strings, que son la palabra “Hola” y el argumento que recibió como parámetro.



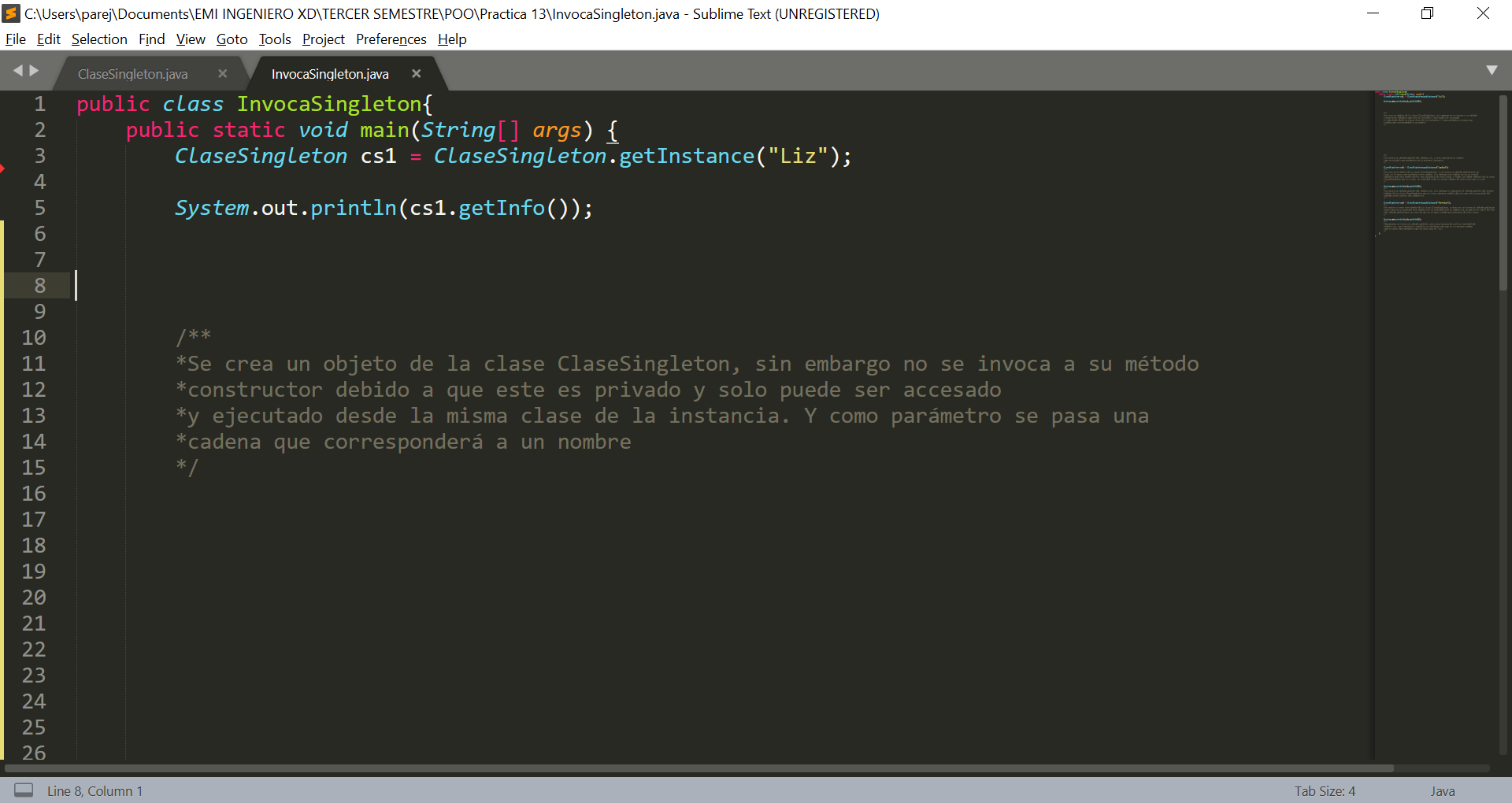
Para poder crear la única instancia de la clase ClaseSingleton desde cualquier otra clase, entonces se necesitará un método público que permita la creación de la única referencia de esta, y este será una sobrecarga del método getInstance, que tendrá como valor de retorno un objeto de la clase ClaseSingleton, que recibe como parámetro una string llamada nombre, dentro de este método en una estructura de control de flujo if, se verificará si el atributo cs, o sea la única instancia de la clase, existe, y si esta si existe, entonces solo se regresará al objeto cs, en caso de no existir, entonces este objeto se crea mediante su método constructor, al cual se le pasa como parámetro la string nombre, y después su valor de retorno será el objeto cs.

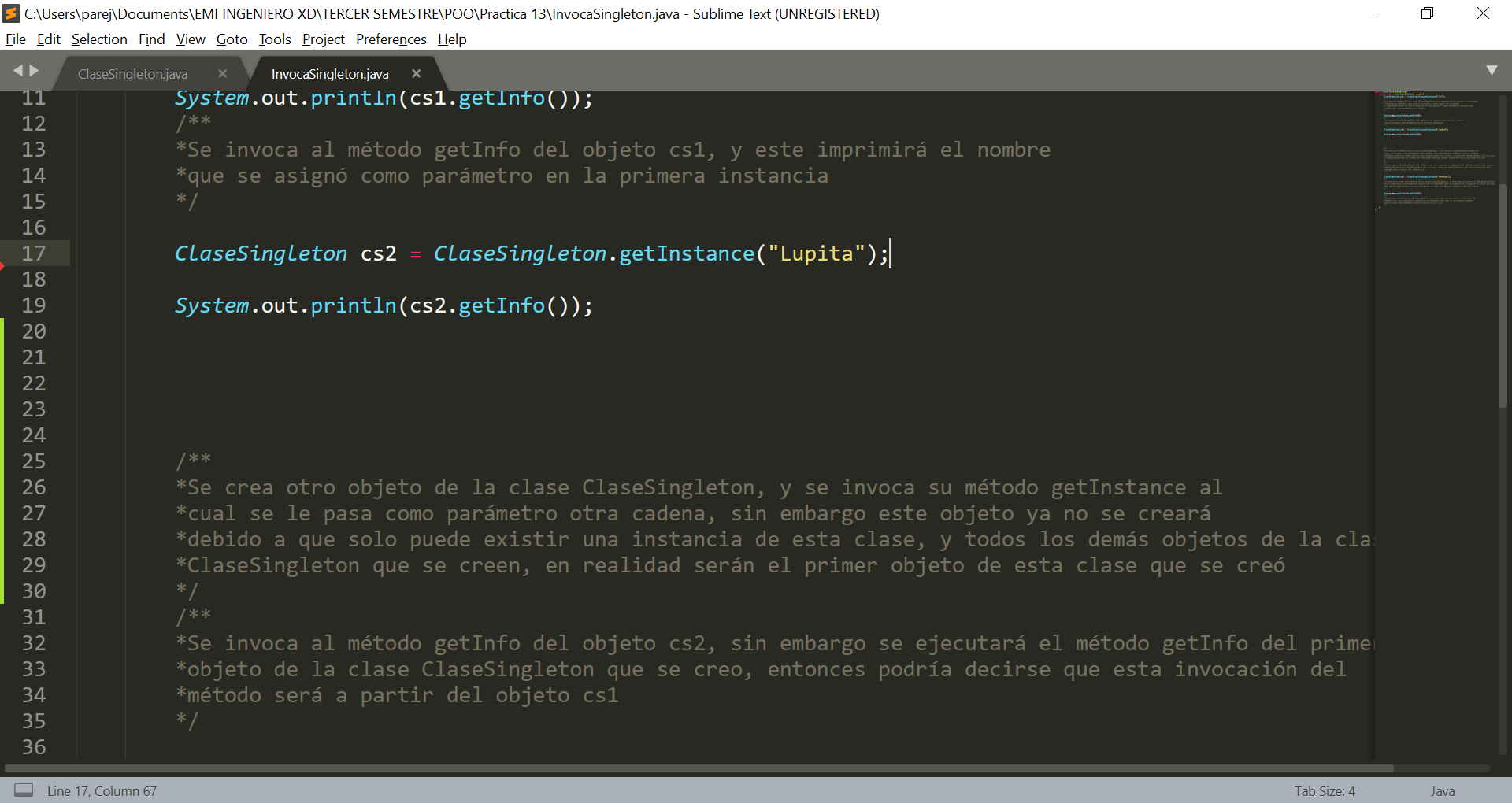


Y finalmente, se instancia su método de tipo string, llamado getInfo, el cual solo tendrá como valor de retorno al atributo info, que será la cadena formada por Hola y un nombre.

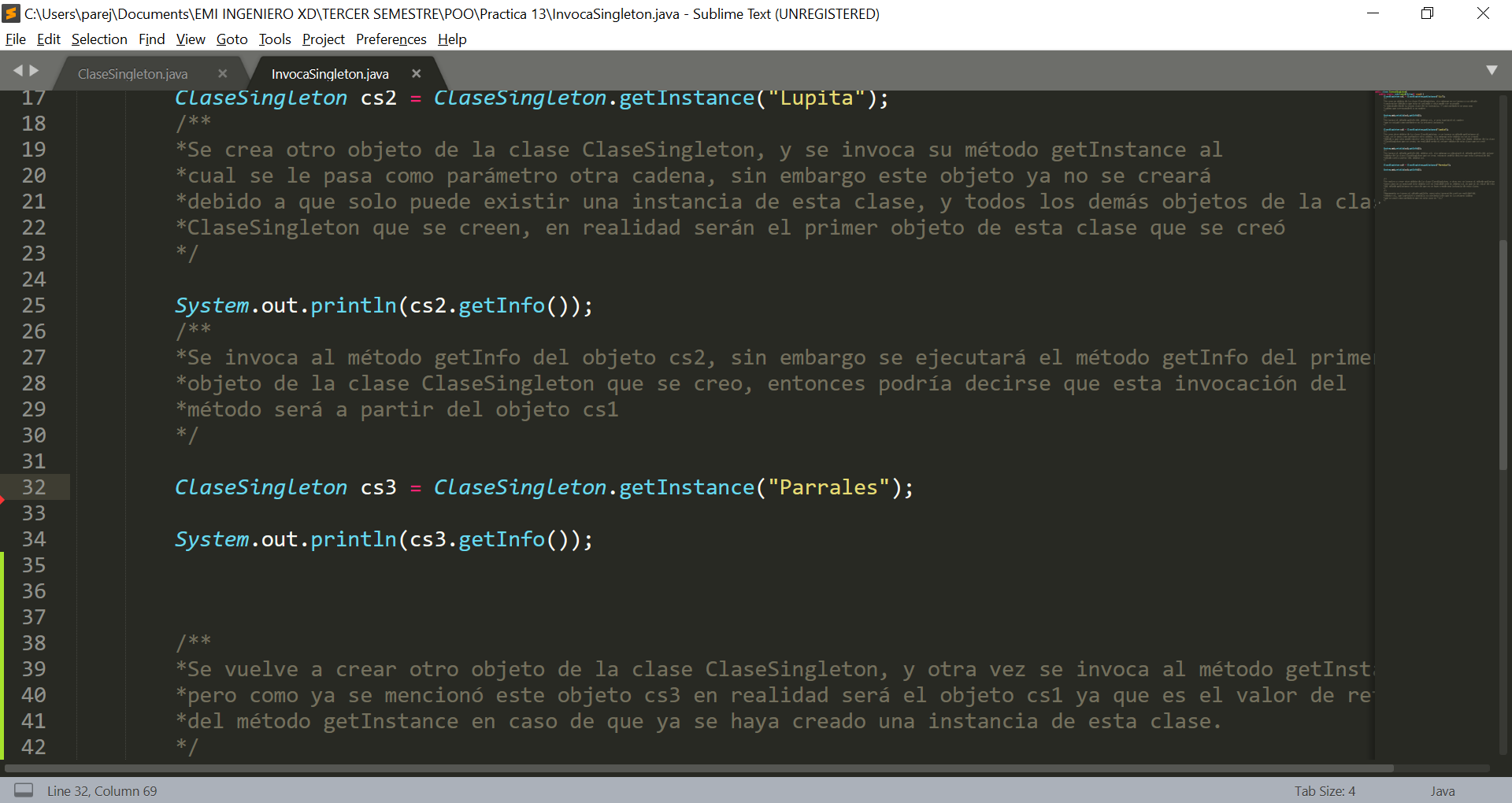
**Clase InvocaSingleton**

En esta clase en su método main, se crea una instancia de la clase ClaseSingleton llamada cs1, y como no se puede invocar a su método constructor, ya que este es privado, entonces se invoca a la función que tiene como valor de retorno una clase ClaseSingleton, que es el método getInstance, y se le pasa como parámetro una string, que en este caso es la palabra “Liz”. Después a partir de este objeto cs1, invocamos a su método getInfo y se imprime a pantalla su valor de retorno que será la cadena “Hola Liz”.

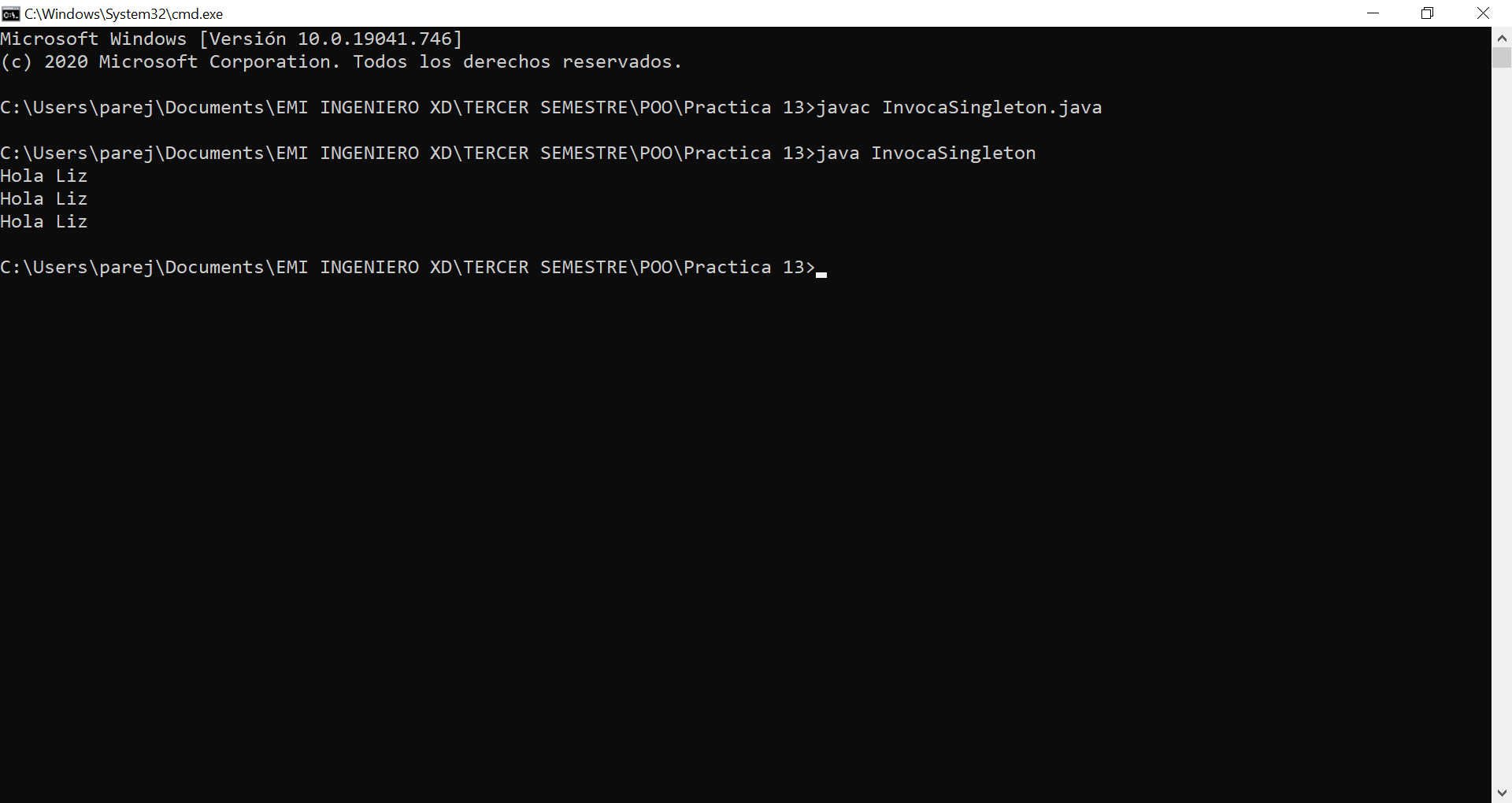
Se crea otra instancia de una clase ClaseSingleton llamada cs2, y también se invoca al método getInstance y se pasa como parámetro a la cadena “Lupita”, sin embargo, como vimos en este método getInstance, si la instancia única de esta clase ya ha sido creada, entonces lo que nos regresará este método será la propia instancia de dicha clase que ya se creó, entonces el objeto cs2, tendrá la referencia del objeto cs1. Y se invoca al método getInfo de este objeto cs2, para que se imprima su valor de retorno, pero lo que se imprimirá a pantalla será el mismo mensaje que apareció anteriormente, ya que no se puede crear otra instancia de esta clase ClaseSingleton, una vez que ya se creó la primera y única.



Lo mismo sucederá con la siguiente instancia que se crea a partir del objeto cs3, al cual al método getInstance, se le asigna como parámetro la cadena “Parrales”, lo que se regrese de este método será la primera instancia de la clase que se creó, por tanto los objetos cs1, cs2 y cs3, serán el mismo, ya que son la única instancia de esta clase que se crea y existe a lo largo de todo el programa. Y el mensaje que se obtenga del método getInfo a partir de la referencia cs3, será el mismo que se obtuvo en las 2 anteriores referencias, “Hola Liz”



* **Ejecución del programa**

**Ejercicio 3**

* **Análisis previo**

La clase InetAddress permite a Java admitir los nombres en internet (url), ya que las direcciones de internet, reducidas a su nivel más bajo, están formadas por un identificador de nodo de 32 bits y un selector de puerto de ese nodo de 32 bits, sin embargo, en la práctica no es conveniente utilizar este esquema de dirección. En internet el direccionamiento lo gestionan los servidores de nombres que traduce nombres habituales y fáciles de recordar a sus correspondientes direcciones de 32 bits.

Esta clase no tiene constructores visibles. Para crear un objeto InetAddress, se tiene que utilizar uno de los métodos disponibles.

Estos métodos son simplemente métodos estáticos que devuelven una instancia de la clase en que residen. Es este caso, InetAddress tiene tres métodos:

* getLocalHost: Devuelve simplemente el objeto InetAddress que representa al nodo local
* getByName: Devuelve un InetAddress que corresponde al nombre de nodo que se le pasa (puede lanzar una excepción de nodo desconocido)
* getAllByName: Devuelve una matriz de InetAddress que representa todas las direcciones en las que se traduce un nombre dado (puede lanzar una excepción de nodo desconocido)

Estos métodos se pueden utilizar para crear instancias de InetAddress.

Las clase InetAddress también tiene cuántos métodos que no son estáticos, que se pueden utilizar sobre los objetos devueltos por los métodos anteriormente mencionados.

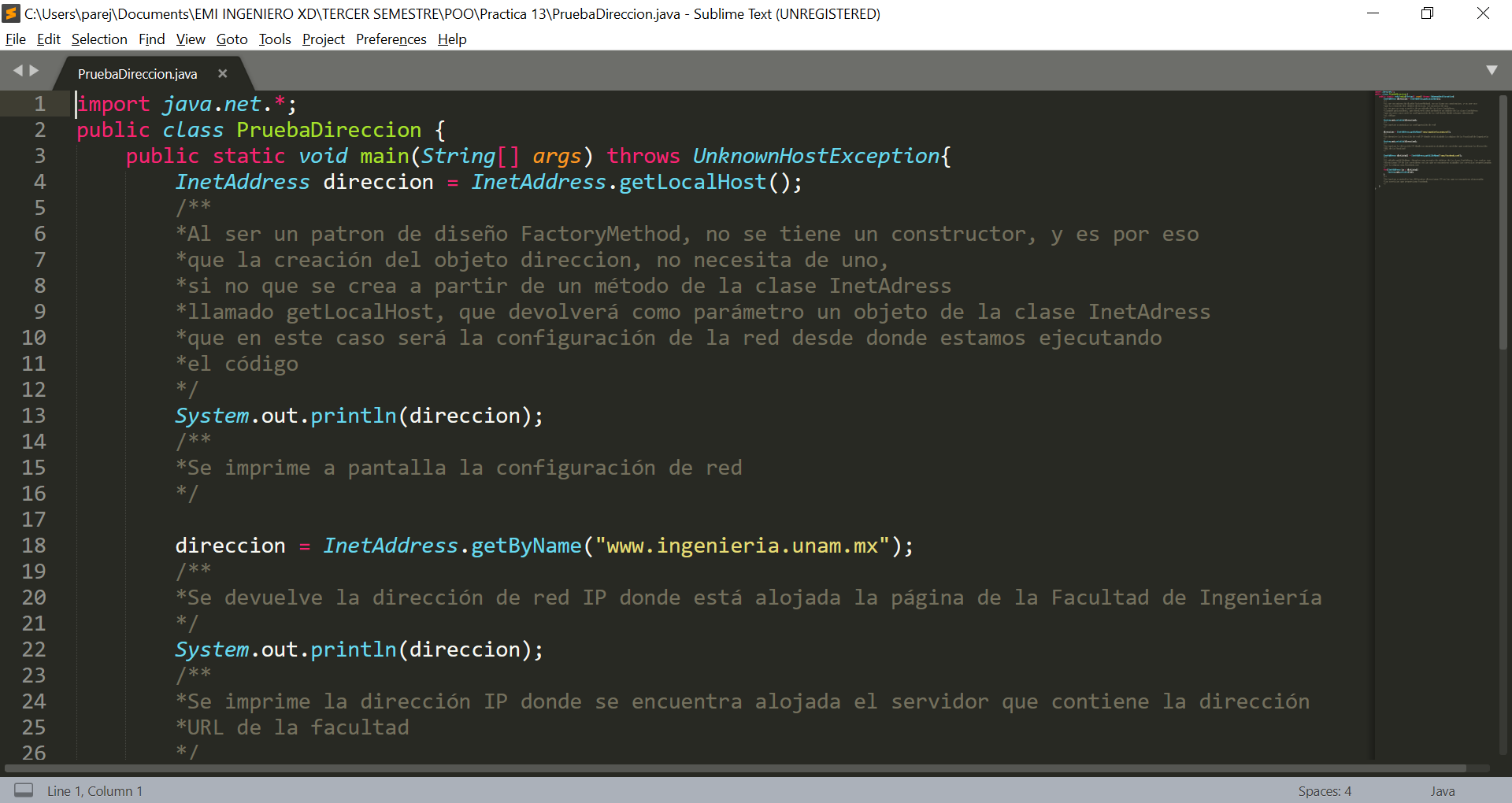
* getHostName(): devuelve una cadena que representa el nombre de nodo asociado con el objeto InetAddress
* getAddress(): devuelve una matriz de bytes de cuatro elementos que representan la dirección en Internet del objeto en el orden de bytes de la red
* toString(): devuelve una cadena que contiene el nombre del nodo y la dirección IP.

Factory Method:

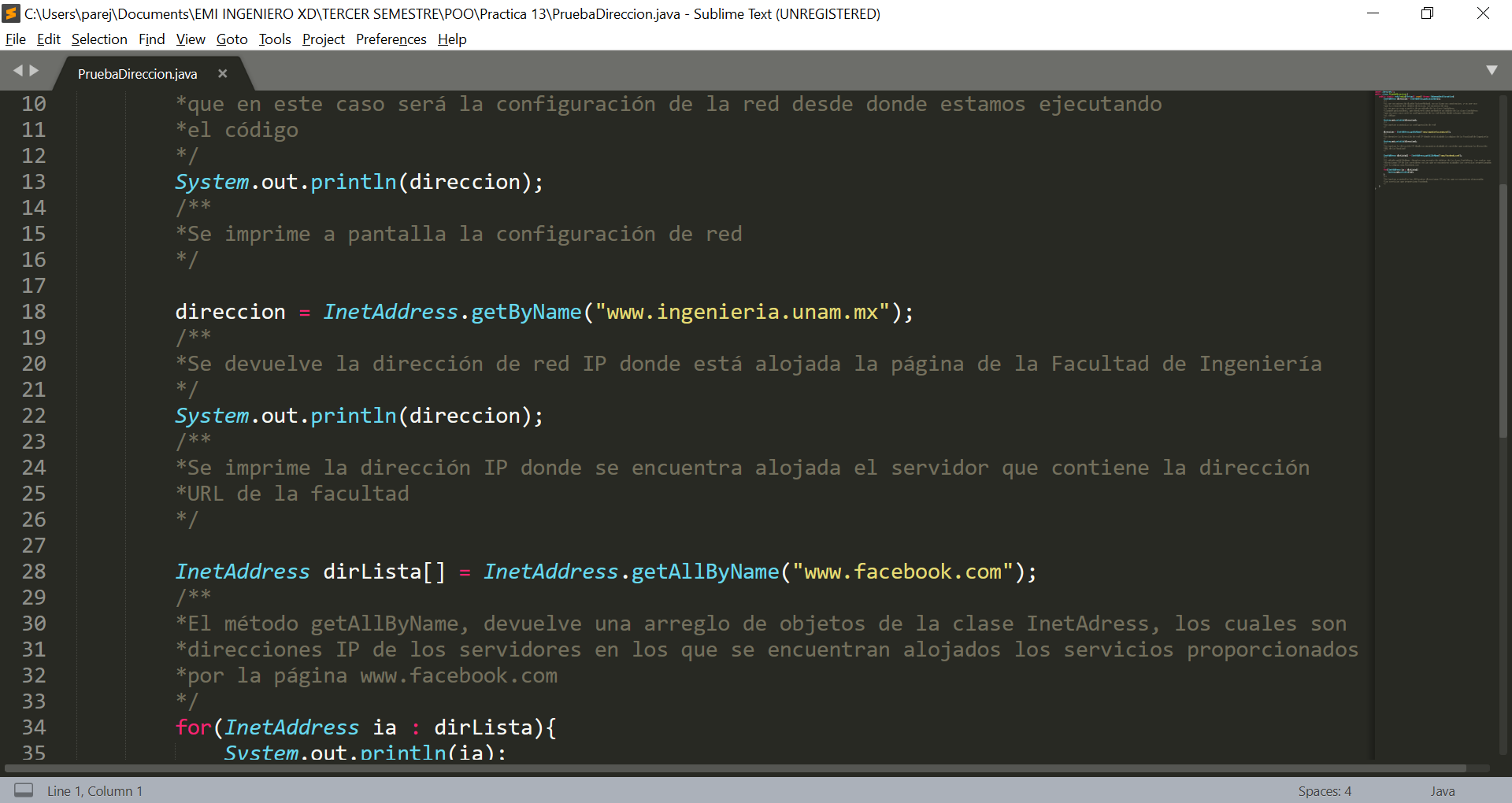
Es un patrón de diseño creacional, este en realidad es un método estático que devuelve un objeto del tipo de la clase que se quiera crear sin la necesidad de un método constructor, y este objeto, puede reutilizarse cuantas veces se quiera para evitar consumir memoria creando más instancias de la clase.

Desarrollo:

En el caso del programa PruebaDireccion, se crea una instancia de la clase InetAddress llamada dirección, a partir de un método de la propia clase InetAddress, el cual es getLocalHost, el cual devolverá la configuración de la red local desde donde se está ejecutando el código. Entonces, se observa que no se necesitó de un método constructor de la clase, para poder generar un objeto de dicha clase. Y una vez que se obtiene la dirección IP de mi servidor, se imprime a pantalla.



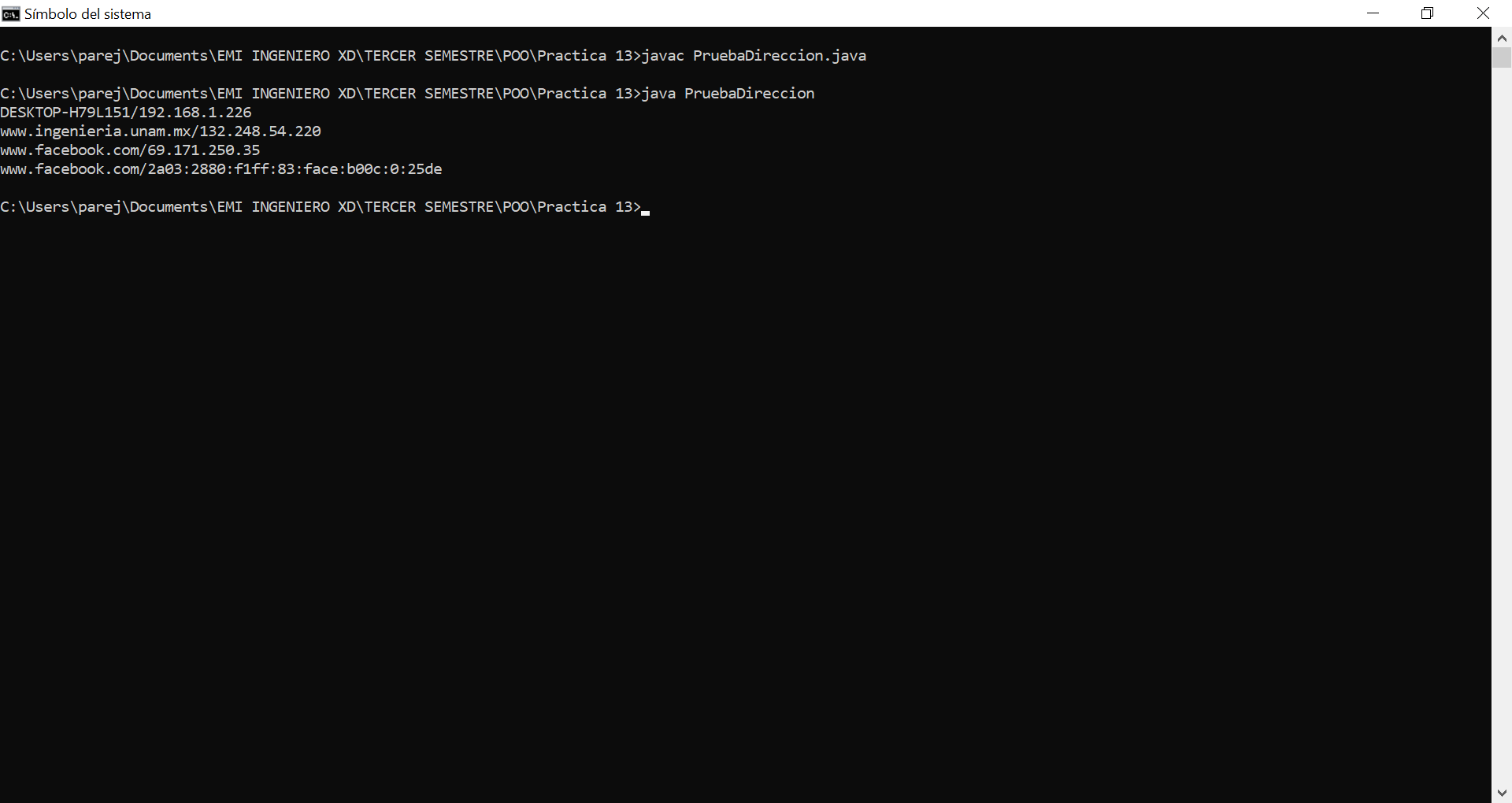
Después se nota que este objeto dirección se puede reutilizar, ya que ahora, nuevamente invocando un método de la clase InetAddress llamado getByName, pasamos como parámetro una dirección URL de la página de la Facultad de Ingeniería y lo que nos devolverá este método es otra dirección IP, en este caso, la del servidor donde se almacenan los servicios que proporciona dicha página, y esta dirección se imprime en pantalla.



Ahora se crea una lista de objetos InetAddress llamada dirLista, y nuevamente no necesitaremos de un método constructor para poder crear el objeto, ya que invocamos al método getAllByName de la clase InetAddress, a la cual se le pasa como parámetro la dirección URL de facebook, y este devolverá todas las direcciones IP de servidores a partir de los cuales facebook proporciona servicios y esta lista en un ciclo for se imprime a pantalla.



* **Ejecución del programa**



**Conclusiones**

* Dávila Ortega Jesús Eduardo:

En el desarrollo de la práctica entendí la importancia de los patrones de diseños así como identificarlos, aparte entendí el manejo de las url para poder obtener direcciones IP, además de como los patrones pueden beneficiar al programador y cómo podemos implementarlos mediante Java.

* Díaz Hernández Marcos Bryan:

En el desarrollo de la práctica aprendí a identificar los patrones de diseño mediante los métodos que están destinados a cumplir con los objetivos de cada uno de estos patrones, como lo es el patrón singleton con el constructor privado y la creación de una única instancia. Además aprendí un poco más de las posibilidades que tiene Java en cuanto al manejo de las bases de datos. Por lo anterior se pudo cumplir con el objetivo de la práctica.

* Pareja Ávila Emiliano:

Se cumplieron los objetivos de la práctica, ya que a partir de los ejemplos de problemas planteados en la práctica y los ejemplos que nos dio la profesora, pude identificar los patrones de diseño más comunes, como el modelo Vista Controlador, el patrón Singleton, el Factory Method y otros más, conozco los problemas que estos patrones pueden resolver y sus aplicaciones, así que cuando se tenga que desarrollar una aplicación, podré analizar las problemas que puedan surgir dentro de la misma, a los cuales quiero dar solución, y aplicar uno o varios patrones de diseño que hagan más sencilla la implementación y ejecución de la aplicación.

* Vázquez Zavala Oliver Alex:

Con la elaboración de esta práctica pude aprender que son los patrones de diseño, como es que nos ayudan para dar solución a un determinado problema que puede presentarse comúnmente al desarrollar aplicaciones similares, de igual manera pude conocer las características de algunos de los patrones de diseño más comunes e implementar una aplicación utilizando un patrón de diseño, aprendí que conocer de patrones de diseño es de gran utilidad, ya que nos permite conocer e implementar las soluciones compartidas por otros desarrolladores a problemáticas recurrentes.

**Bibliografía**

* Gamma Erich, Helm Richard, Patrones de diseño-Software Orientado a objetos, Addison Wesley, México 2002.