|  |  |  |
| --- | --- | --- |
|  | **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):* | 12 |
| *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:* | 19 de enero del 2021 |
| *Observaciones:* |  |
|  |  |

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_

**Objetivo de la práctica.**

Implementar el concepto de multitarea utilizando hilos en un lenguaje orientado a objetos.

**Introducción.**

En la actualidad la mayoría de los sistemas operativos proporcionan características que permiten que un proceso (un programa en ejecución), tenga múltiples hilos de control. Un hilo (Thread) es la ejecución de ciertas instrucciones dentro de un proceso, por lo cual podemos decir que un proceso está conformado por uno o varios hilos, al trabajar con hilos es posible tener múltiples flujos en ejecución sin la necesidad de que una instrucción (hilo) dependa de la ejecución de otras instrucciones, permitiendo realizar multitareas en un determinado programa, así por ejemplo un usuario puede estar usando una aplicación mientras otra tarea se encuentra ejecutándose a la par, los hilos tienen un ciclo de vida, pasando por diferentes estados, los cuales son:

-New: El primer estado de un hilo recién creado, permanece en este estado hasta que el hilo es ejecutado.

-Runnable: Una vez ejecutado pasa a este estado, durante el cual ejecuta su tarea.

-Not runnable: Estado que permite al hilo desocupar la CPU en espera a que otro hilo termine o le notifique que puede continuar, o bien a que termine un proceso de entrada/salida o incluso debido a que termine una espera provocada por algún método como “suspend” o “sleep” y tras ello volverá al estado Runnable.

-Dead: Un hilo pasa al estado de muerto cuando se termina su bloque de instrucciones o en cualquier momento al llamar a su método “stop”.

Además de lo anterior un hilo posee un atributo conocido como prioridad (un valor entre 1 y 10), dicha prioridad de un hilo determina, en parte, cuánto tiempo de CPU recibe un hilo en relación con los otros hilos activos, cuando se crea un hilo generalmente este posee una prioridad de 5 (a menos de que se especifique otra al momento de su creación). Para la implementación de hilos en java estos están encapsulados en la clase Thread, por lo tanto, para crear un hilo se tienen dos posibilidades:

-Heredar de Thread redefiniendo el método run().

-Crear una clase que implemente la interfaz Runnable que nos obliga a definir el método run().

En ambos casos debemos definir un método run() que será el que contenga el código del hilo. Desde dentro de este método podremos llamar a cualquier otro método de cualquier objeto, pero este método run() será el método que se invoque cuando iniciemos la ejecución de un hilo. El hilo terminará su ejecución cuando termine de ejecutarse este método run() por ende el hilo pasará al estado de muerto.

Para crear nuestro hilo mediante herencia se hace de la siguiente manera:

*public class Hilo extends Thread*

*{*

*public void run(){*

*// Código del hilo*

*}*

*}*

Una vez definida la clase de nuestro hilo deberemos ejecutarlo de la siguiente forma:

*new Hilo().start();*

Al llamar al método start del hilo, comenzará a ejecutarse su método run. Crear un hilo heredando de Thread tiene el problema de, al no haber herencia múltiple en Java, si se hereda de Thread no podremos heredar de ninguna otra clase, y por lo tanto un hilo no podría heredar de ninguna otra clase., este problema desaparece si utilizamos la interfaz Runnable para crear el hilo, ya que una clase puede implementar varias interfaces. Definiremos la clase que contenga el hilo como se muestra a continuación:

*public class Hilo implements Runnable*

*{*

*public void run()*

*{*

*// Código del hilo*

*}*

*}*

Para ejecutar un hilo de este tipo deberemos hacer lo siguiente:

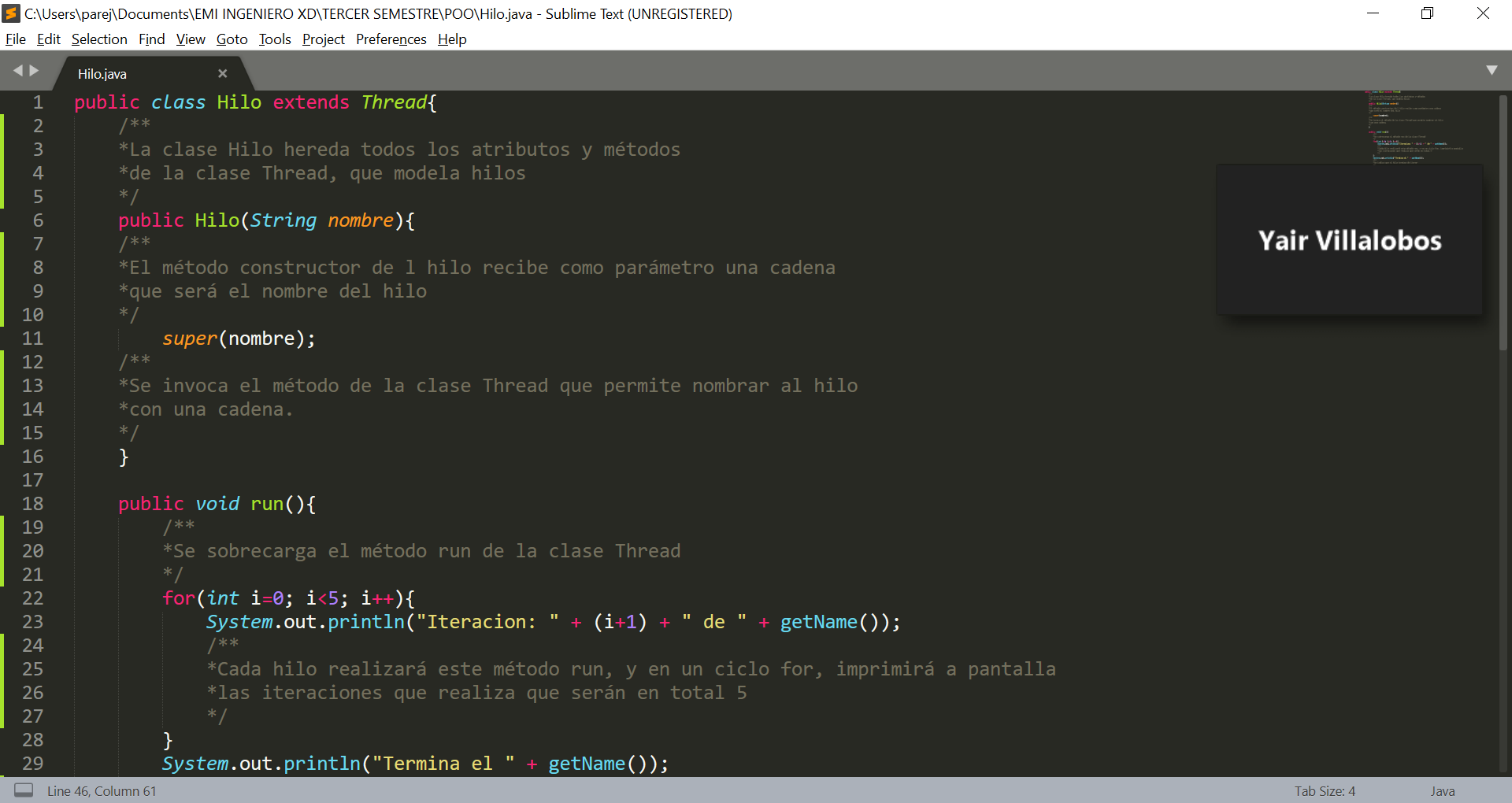
*new Thread(new EjemploHilo()).start();*

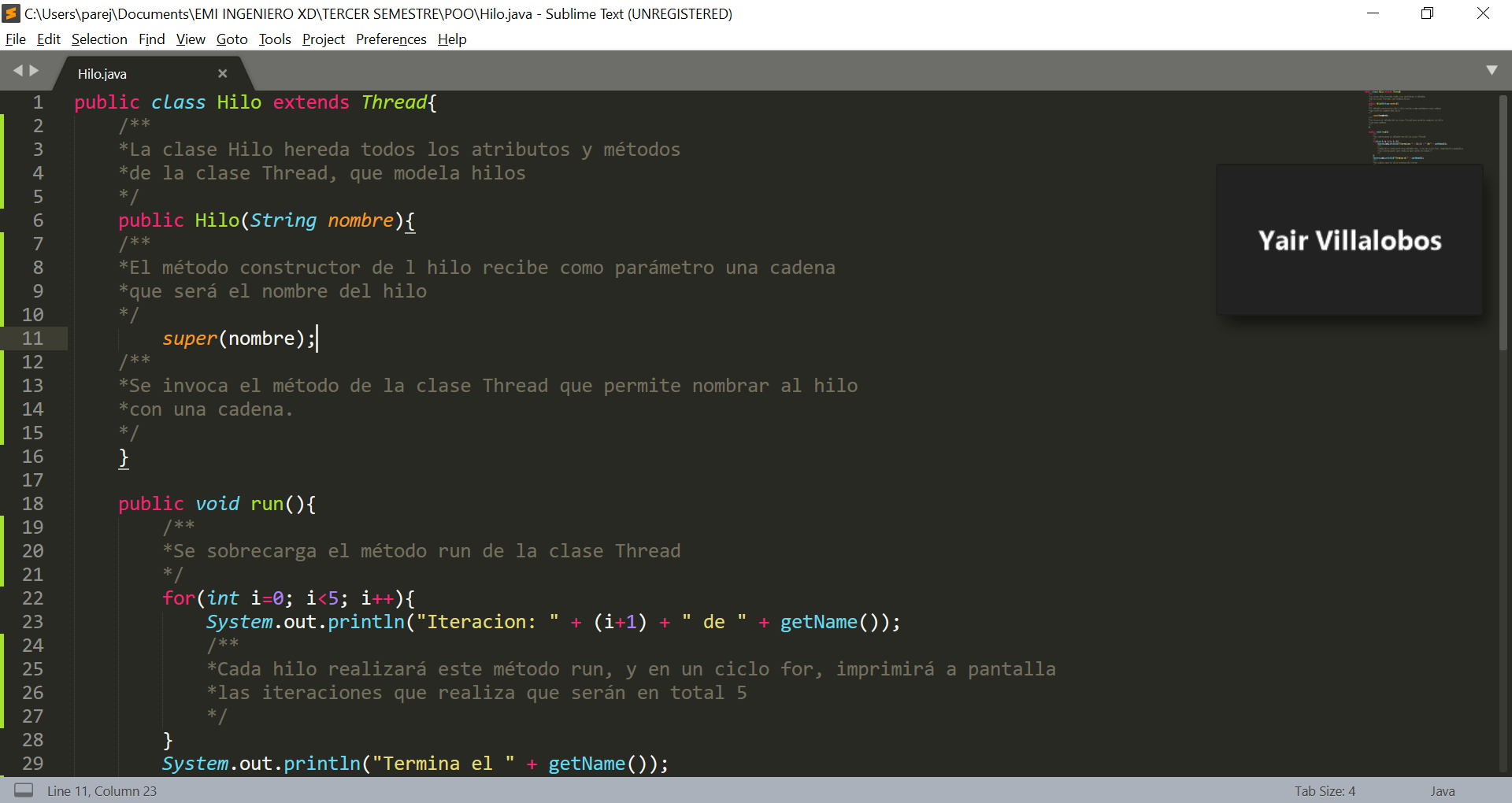
Esto es así debido a que en este caso *Hilo* no deriva de una clase Thread, por lo que no se puede considerar un hilo, lo único que estamos haciendo al implementar la interfaz es asegurar que vamos a tener definido el método run(). Con esto lo que haremos será proporcionar esta clase al constructor de la clase Thread, para que el objeto Thread que creemos llame al método run() de la clase que hemos definido al iniciarse la ejecución del hilo, ya que implementando la interfaz le aseguramos que esta función existe.

**Análisis de los ejercicios.**

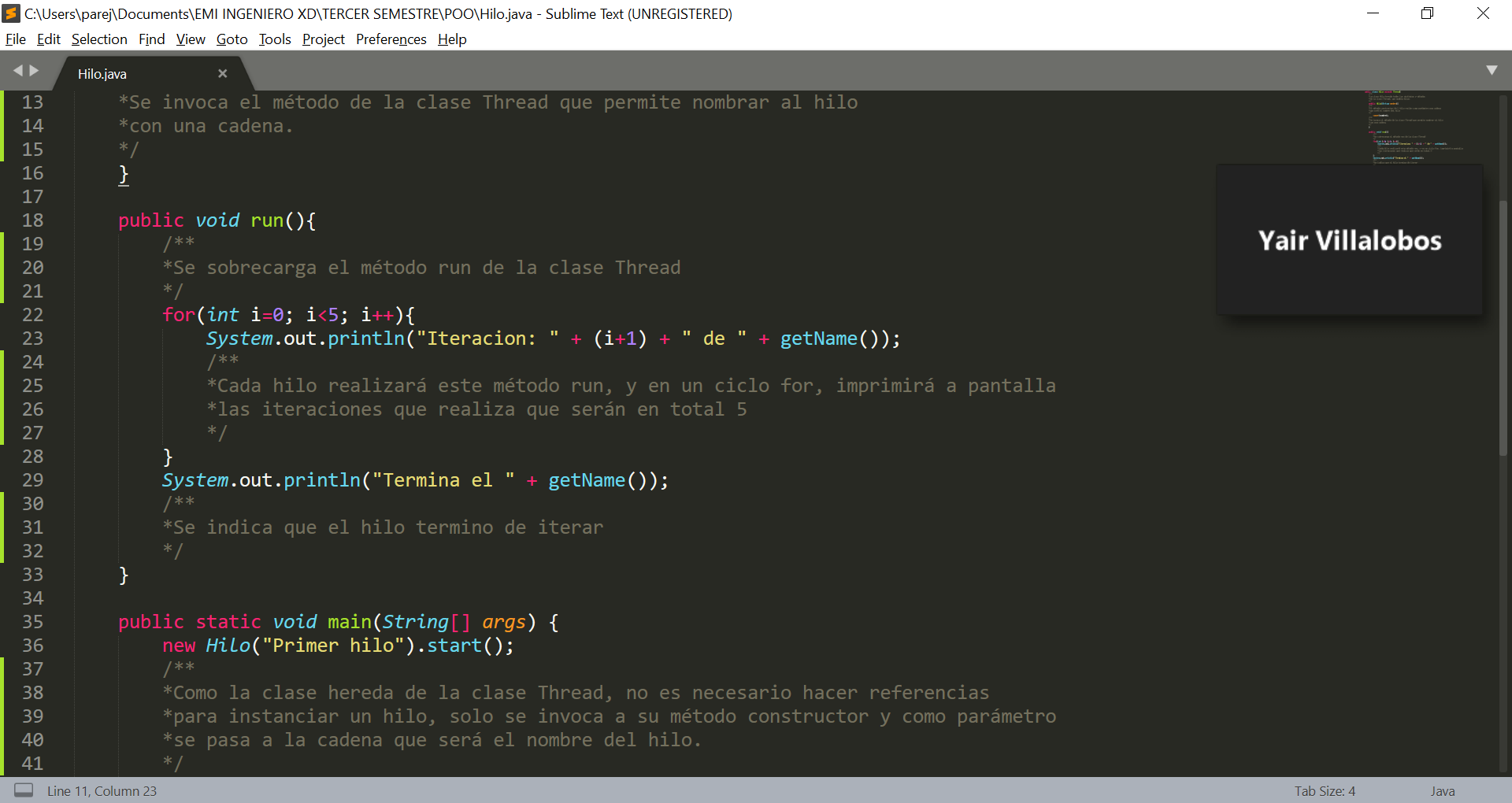
* **Análisis-Hilo.java**

En el primer ejemplo, se crea una clase llamada Hilo, que hereda de la clase Thread, lo que significa que heredará sus atributos y métodos, además como se vio en clase, esta es la manera más fácil de poder implementar los hilos.



Después en su método constructor, recibe como parámetro una cadena, que será el nombre del hilo. Y se invoca al método de la clase Thread, que permite nombrar al hilo.

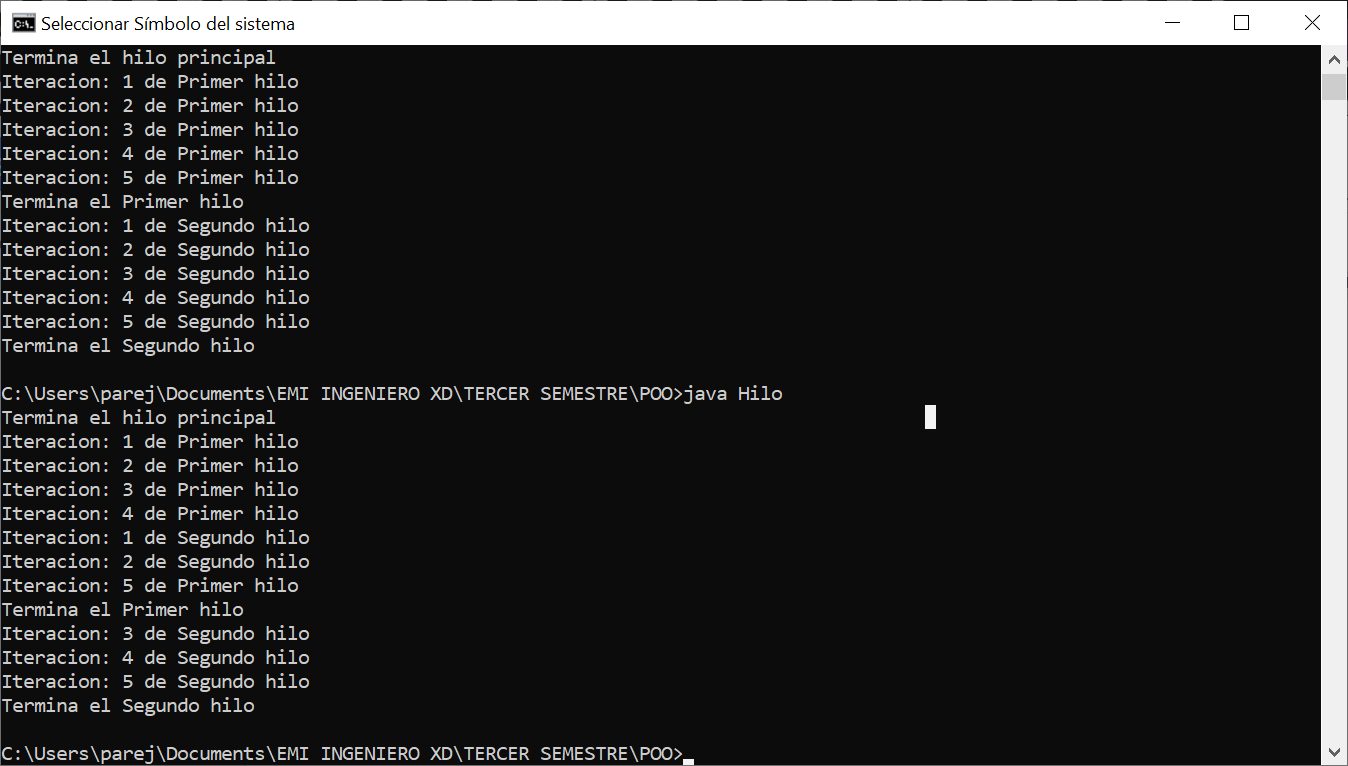
Después fuera del método constructor, se hace una sobrecarga del método run, que será invocado por el método start en el método main de la misma clase, y cada hilo que se cree y que invoque el método start, realizará las instrucciones del método run. En este método run, en un ciclo de repetición for, se imprimen a pantalla las iteraciones que realiza cada hilo, y al finalizar este ciclo for, se imprime a pantalla que el hilo ha acabado de iterar.

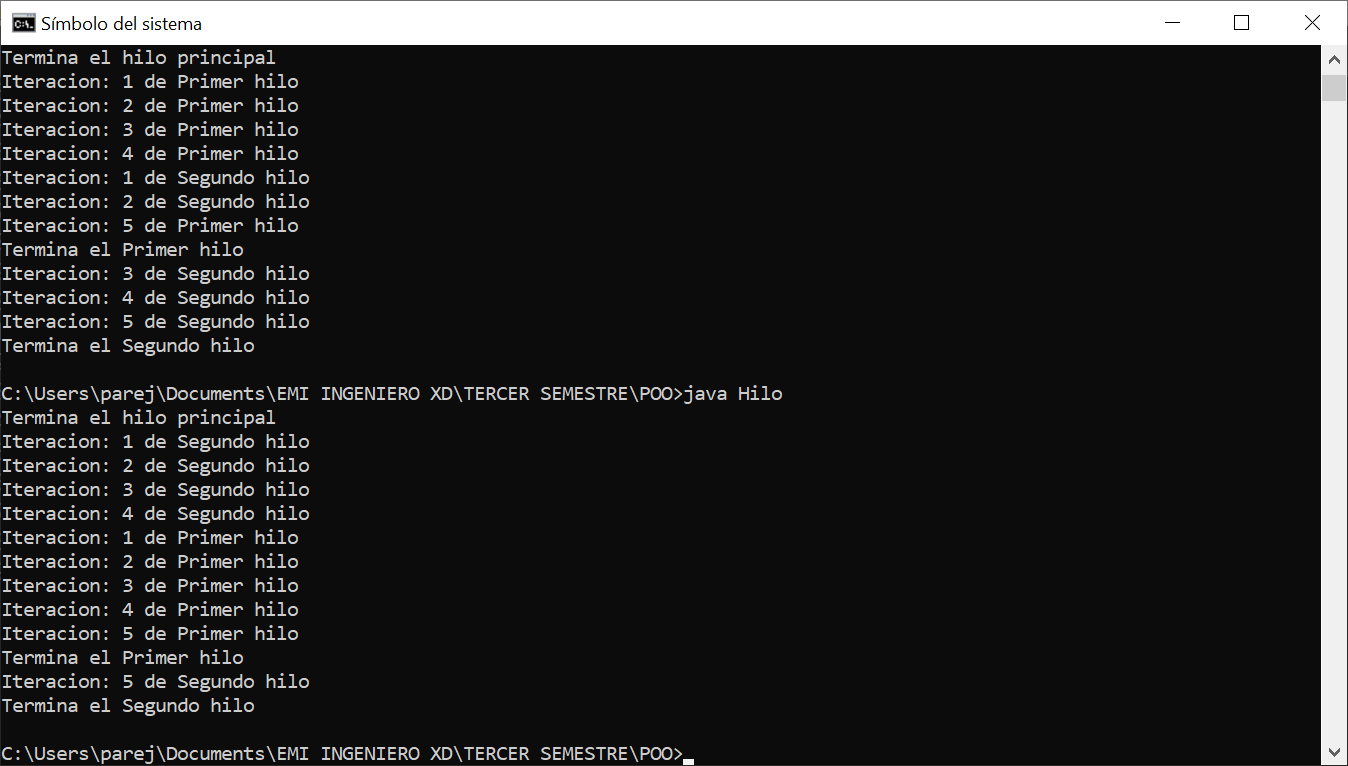
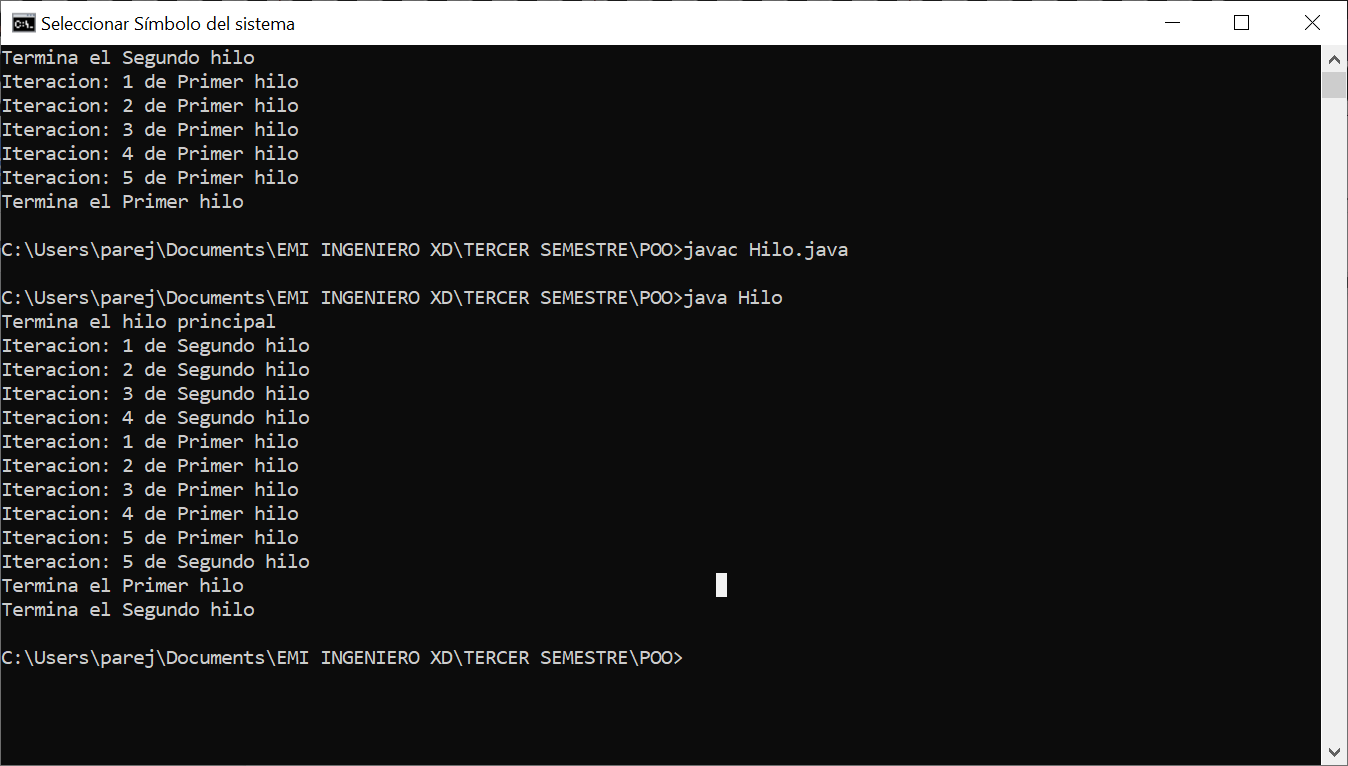


Después se instancia el método main, y en este se crean dos hilos, de la clase Hilo, y para poder crear estos, no es necesario hacer una referencia de la clase Hilo, ya que esta clase Hilo hereda de la clase Thread y solo se crean los hilos mediante su método constructor, el cual tendrá como parámetro el nombre del hilo, y posteriormente se invoca el método start del hilo, que a su vez invocará al método run de dicho hilo. En este caso se crean dos hilos que reciben como nombre “Primer Hilo” y “Segundo Hilo” y se invocan sus métodos start. Después en una impresión a pantalla, se muestra el mensaje que la ejecución del hilo principal ha acabado, pero esto no significará que al terminar el hilo principal, los hilos que se crearon dentro de él, terminarán su ejecución, ya que son 2 procesos totalmente independientes.



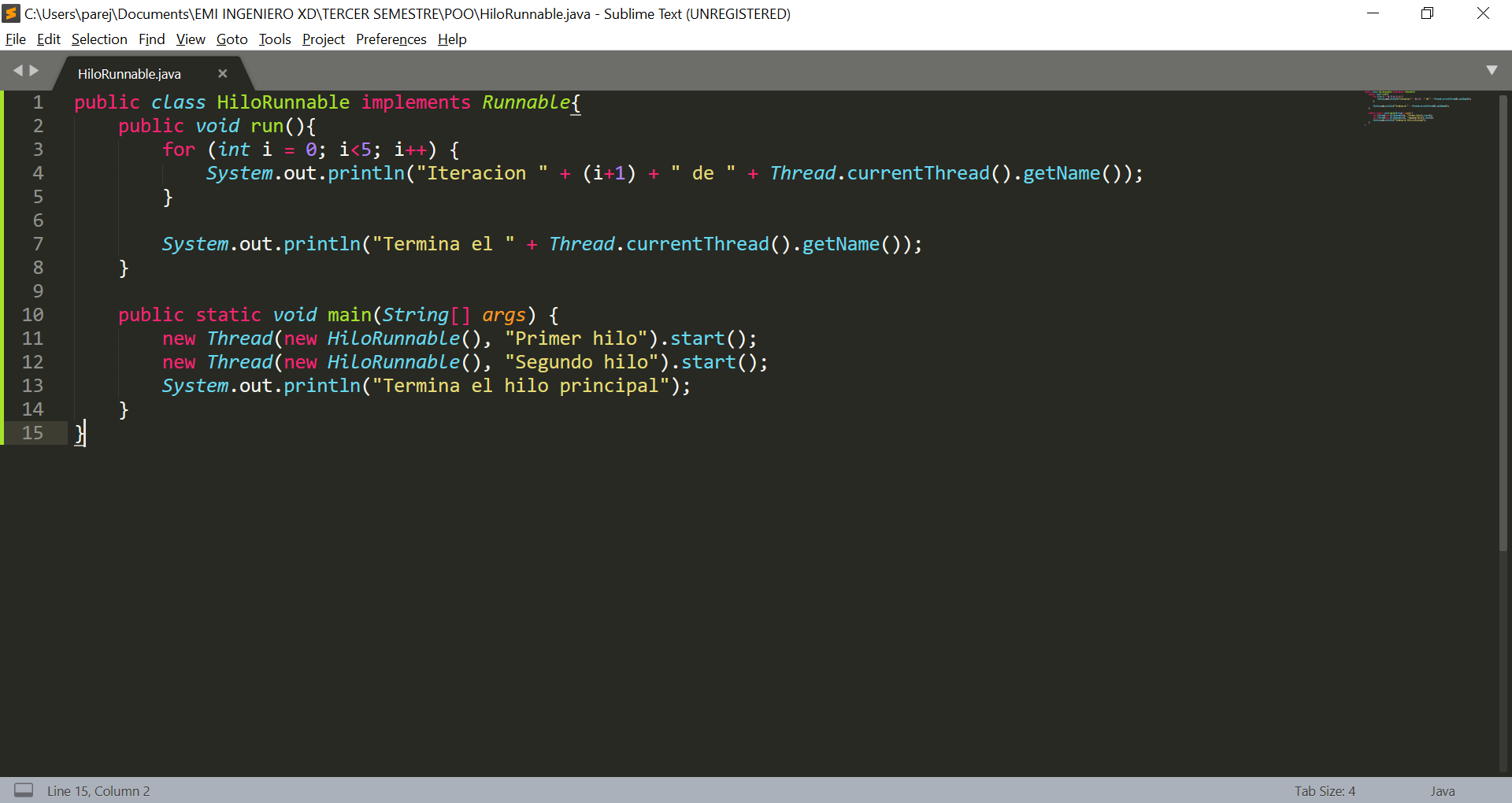
* **Ejecución del programa**

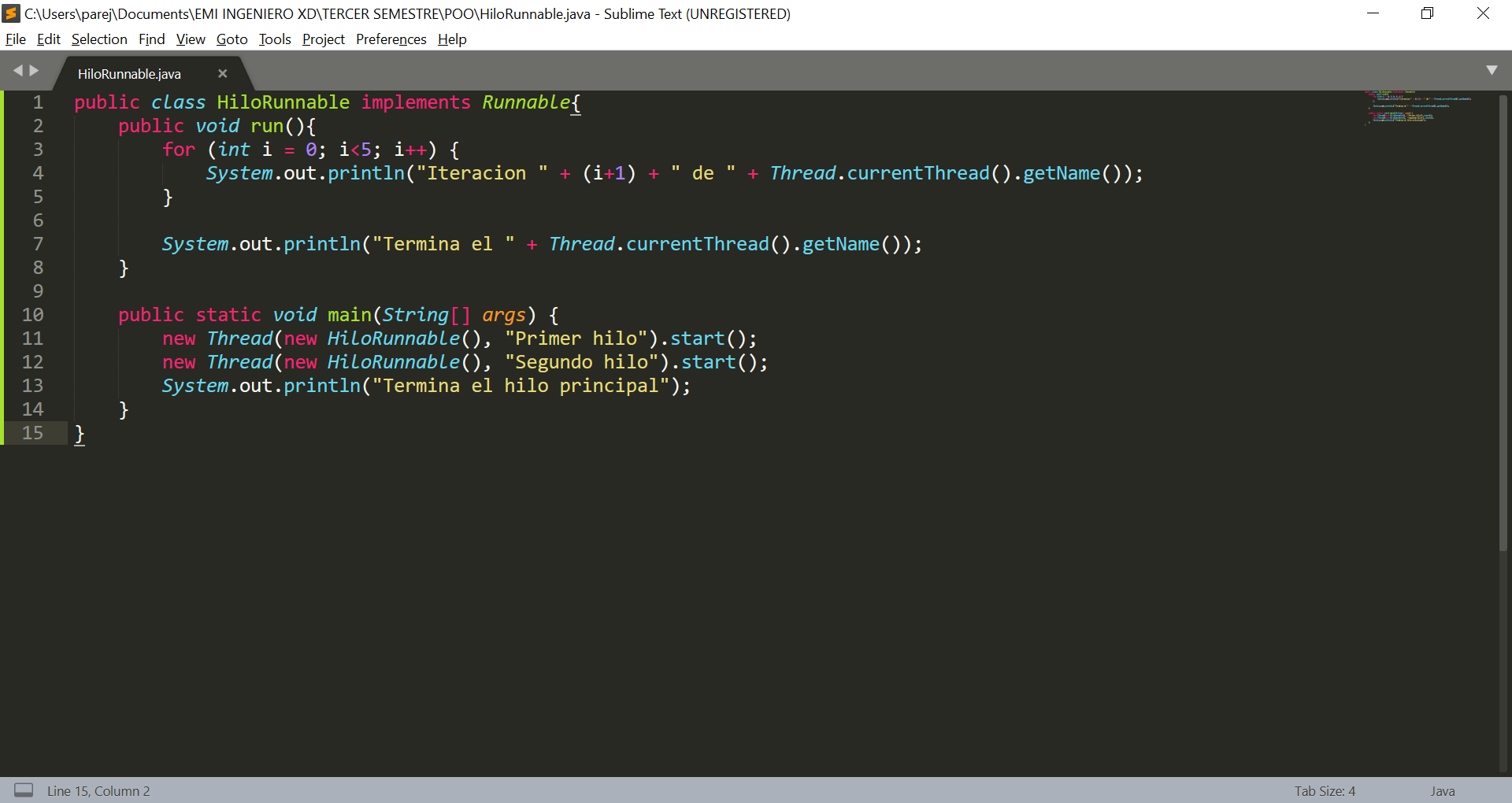
Corrimos varias veces el programa, ya que siempre darán salidas diferentes, esto debido a que los hilos al tener una prioridad media de 5, estarán compitiendo por la los recursos simultáneamente, y las impresiones a pantalla de sus iteraciones serán aleatorias y sin ningún orden.

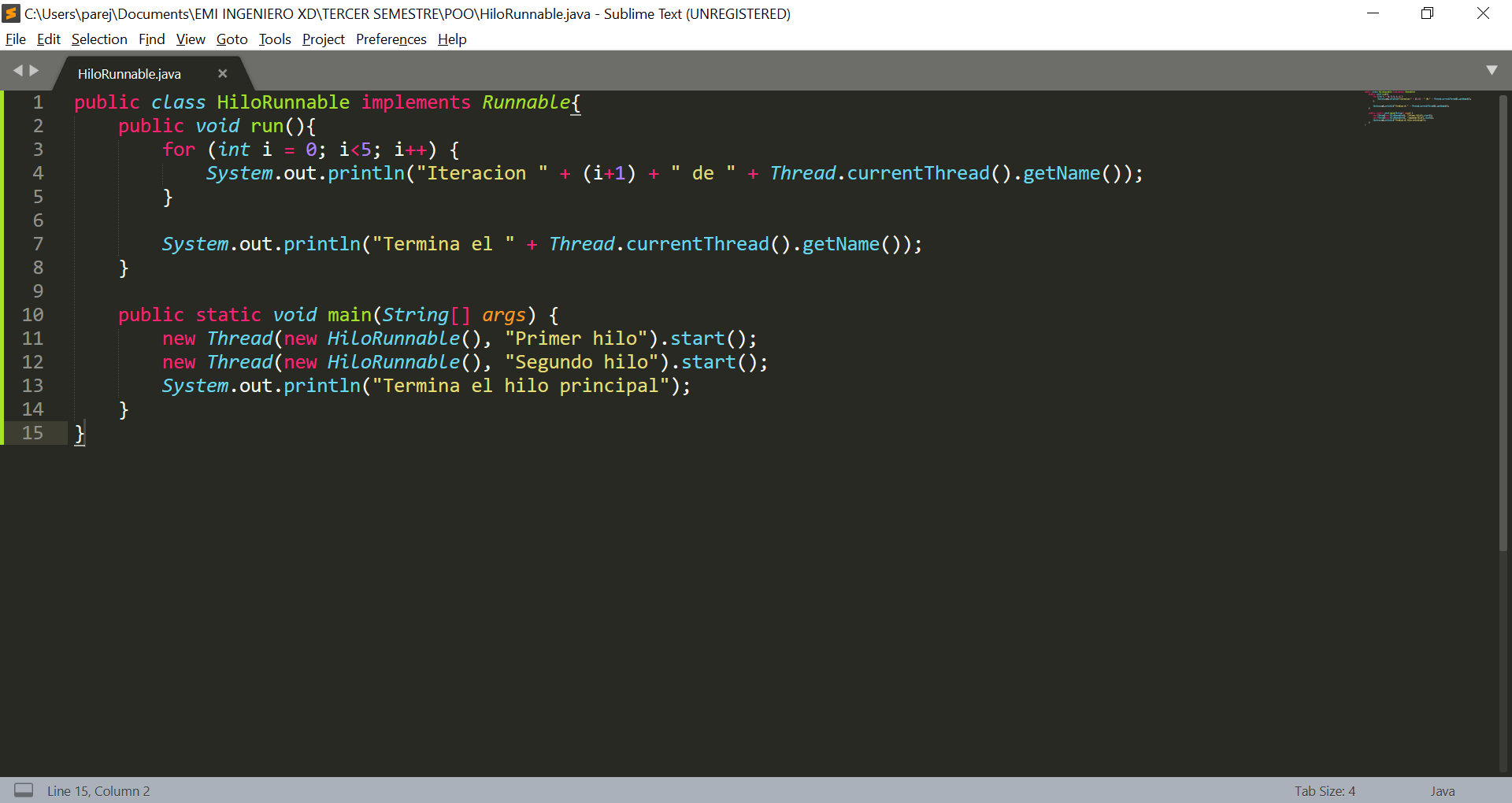


* **Análisis-HiloRunnable.java**

Este programa es bastante similar al anterior, solo que en la creación de la clase HiloRunnable, indicamos que la clase implementará la interfaz Runnable, dicha interfaz es implementada por la clase Thread, y se recomienda utilizarla si solo se sobrecargará el método run, ya que es más eficiente y fácil, trabajar hilos que hereden directamente de la clase Thread.

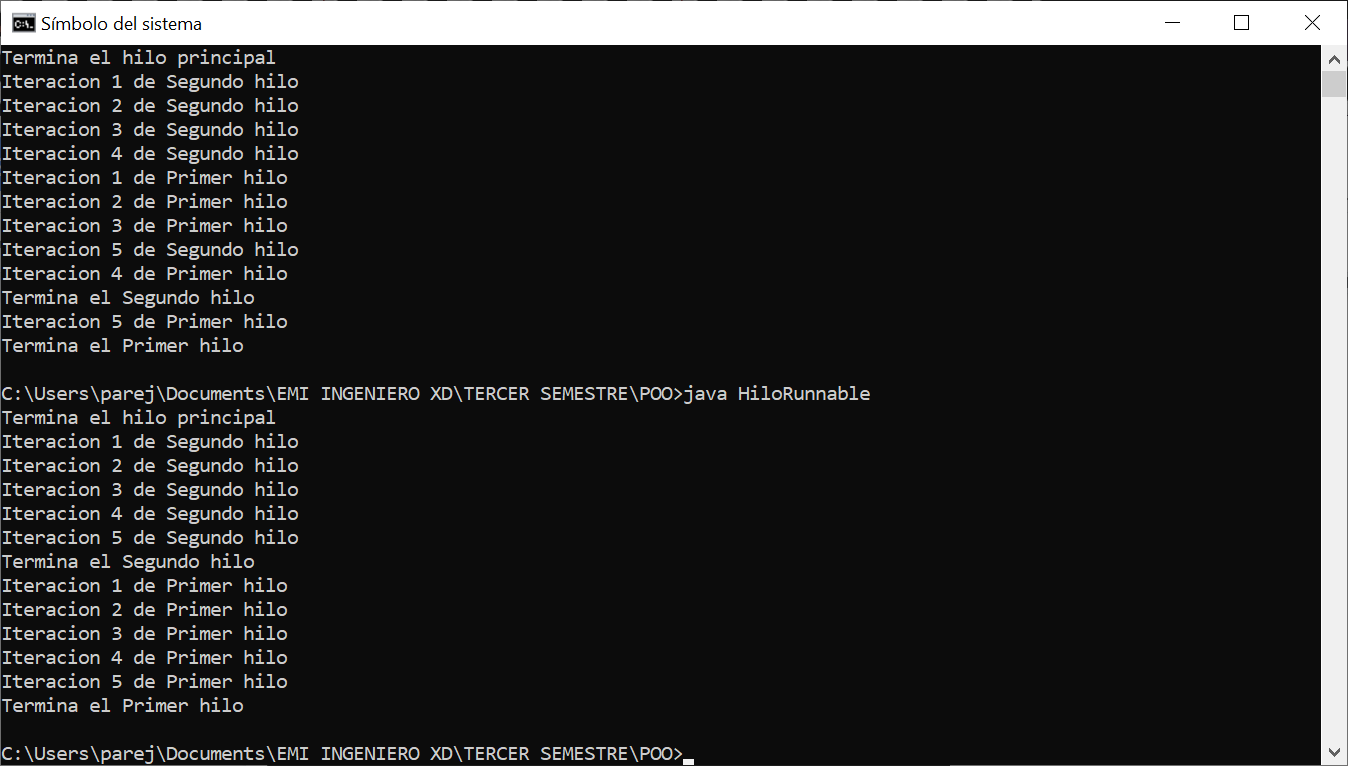
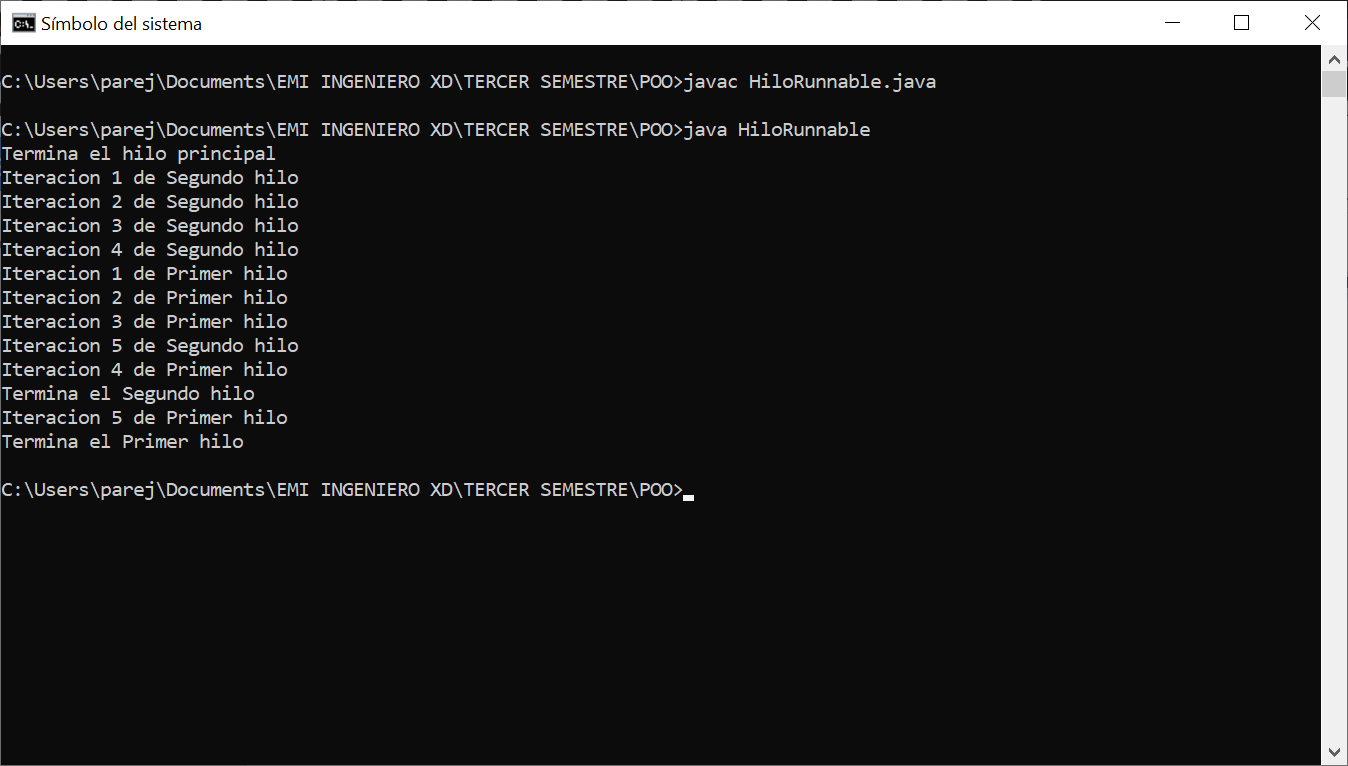
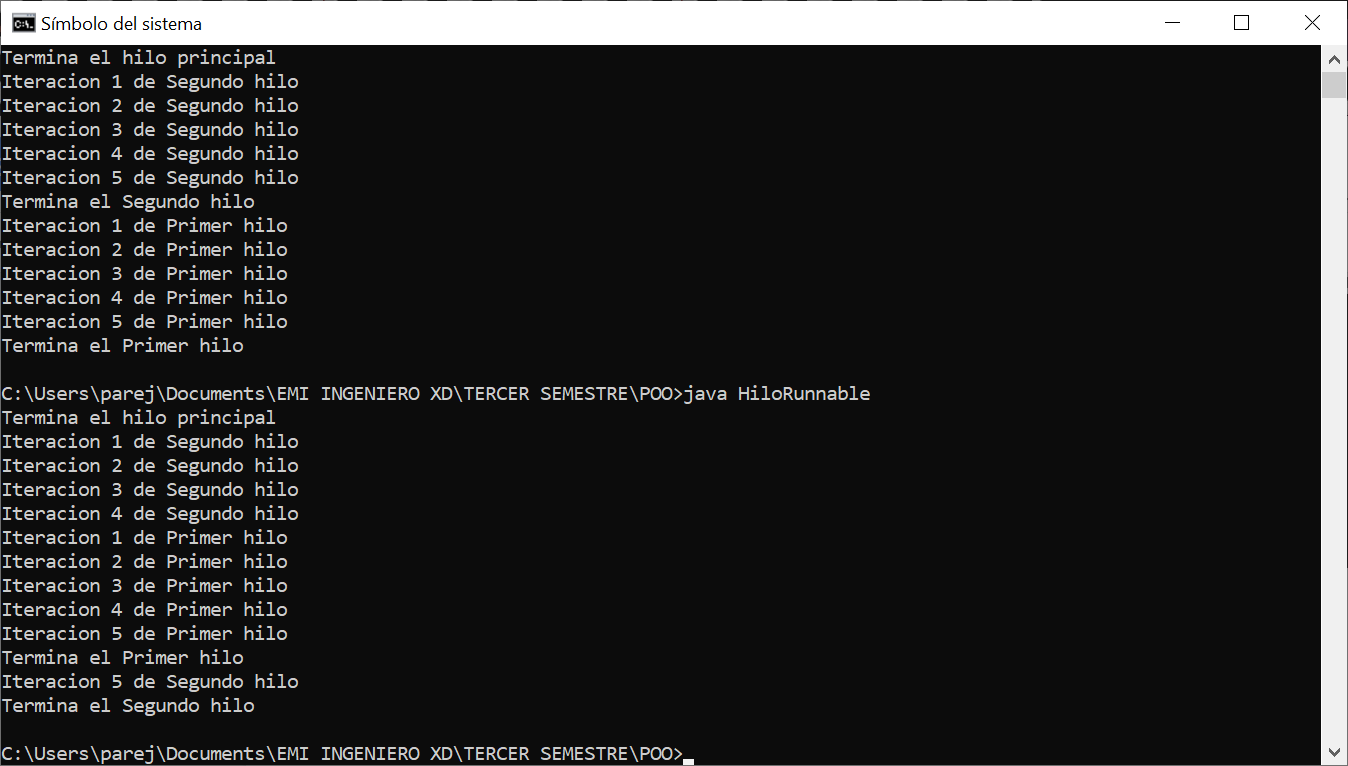
Como vemos, en este caso no se define un método constructor, ya que en el método main, utilizaremos uno de los métodos constructores predeterminados de la clase Thread, para crear el hilo. Después se hace una sobrecarga del método run de la propia interfaz Runnable, que hará lo mismo que el ejercicio anterior, en un ciclo for que itera de i=0, hasta i=4, se imprimirá a pantalla el número de iteración y el hilo que está realizando dichas iteraciones, en este caso para obtener el nombre del hilo, mediante un objeto de la clase Thread, se hace una referencia del hilo actual que está ejecutando el método run con ayuda del método currentThread, y de ese hilo, obtenemos su nombre con su método getName. Al finalizar las iteraciones de dicho hilo, se imprime a pantalla que el hilo que realizó el método run acabó su ejecución.

Después en el método main de la clase HiloRunnable, y para crearlos, no se necesita una referencia de la clase, entonces se crean dos hilos mediante un método constructor de la clase Thread, que recibe como parámetro, el objeto al cual hará referencia el hilo, y el nombre del hilo, en este caso, los dos objetos al cual harán referencia los dos hilos creados, será uno de la clase HiloRunnable, y como nombres, se les asigna “Primer Hilo” y “Segundo Hilo”. Después se hace la invocación al método start, que a su vez invocará al método run de la clase Thread y que realizará el método run definido dentro de la clase HiloRunnable. Una vez que se crean y los dos hilos están en ejecución, se imprime a pantalla que la ejecución del hilo principal terminó, sin que esto signifique que también los hilos que se crearon terminen su ejecución.



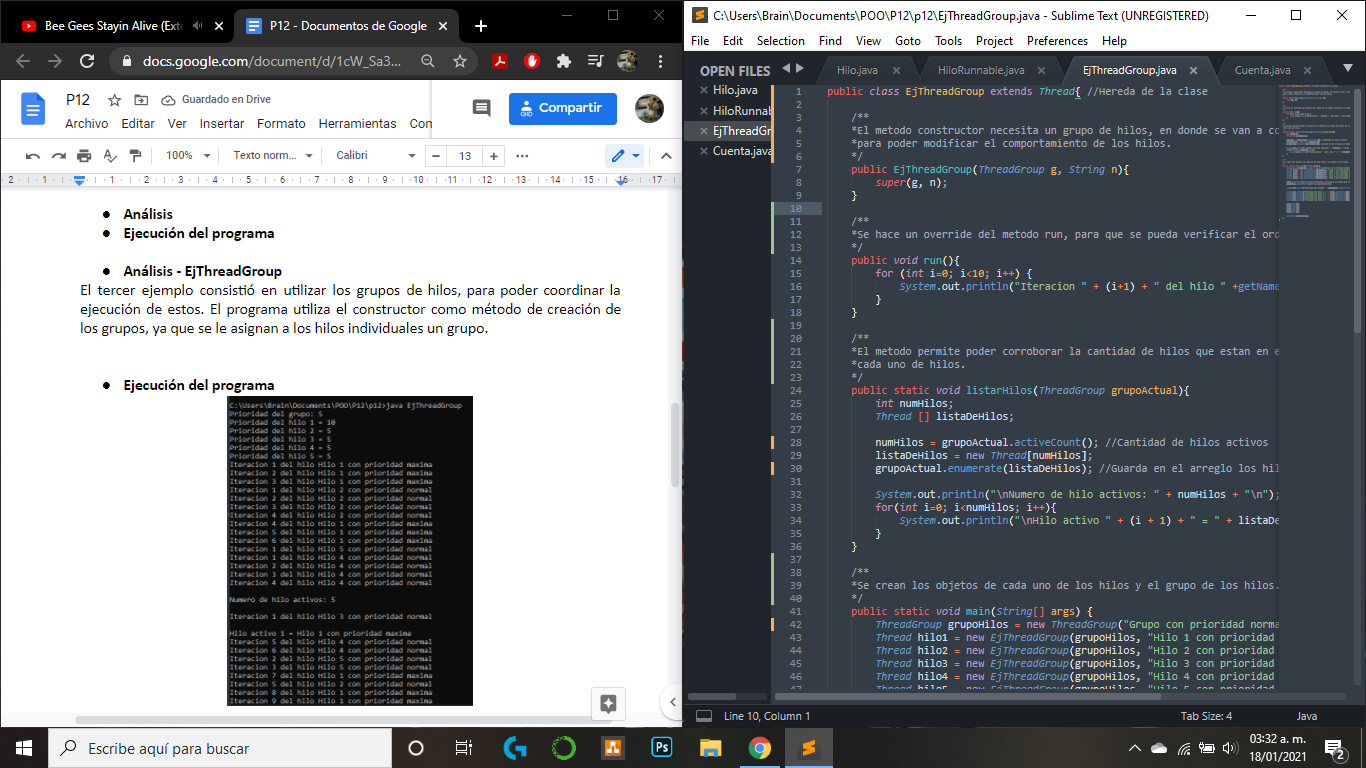
* **Ejecución del programa**

Al igual que en el ejercicio anterior, realizamos varias pruebas de este algoritmo, ya que siempre darán salidas aleatorias, puesto a que la prioridad de ambos hilos creados es de 5, lo cual hará que estos estén compitiendo por los recursos, y den salidas sin ningún tipo de orden.



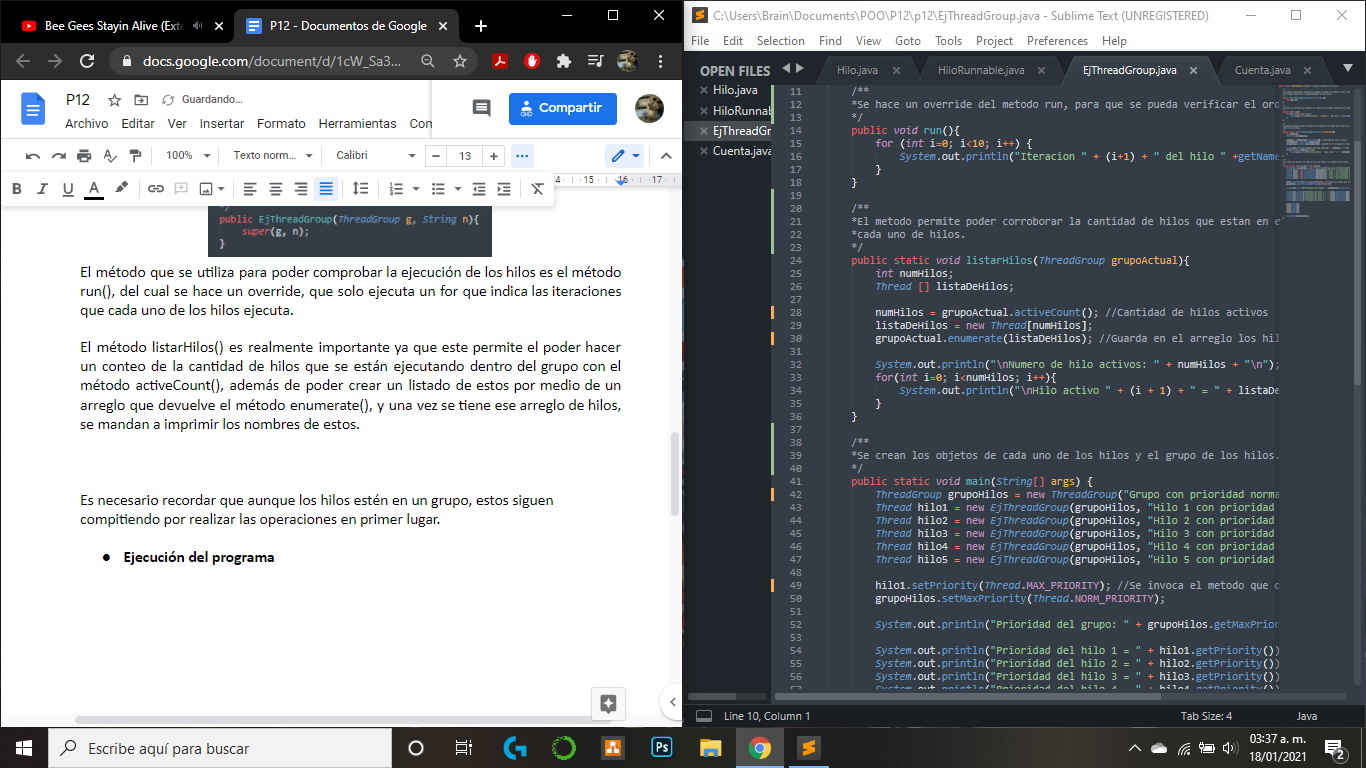
* **Análisis - EjThreadGroup**

El tercer ejemplo consistió en utilizar los grupos de hilos, para poder coordinar la ejecución de estos. El programa utiliza el constructor como método de creación de los grupos, ya que se le asignan a los hilos individuales un grupo.

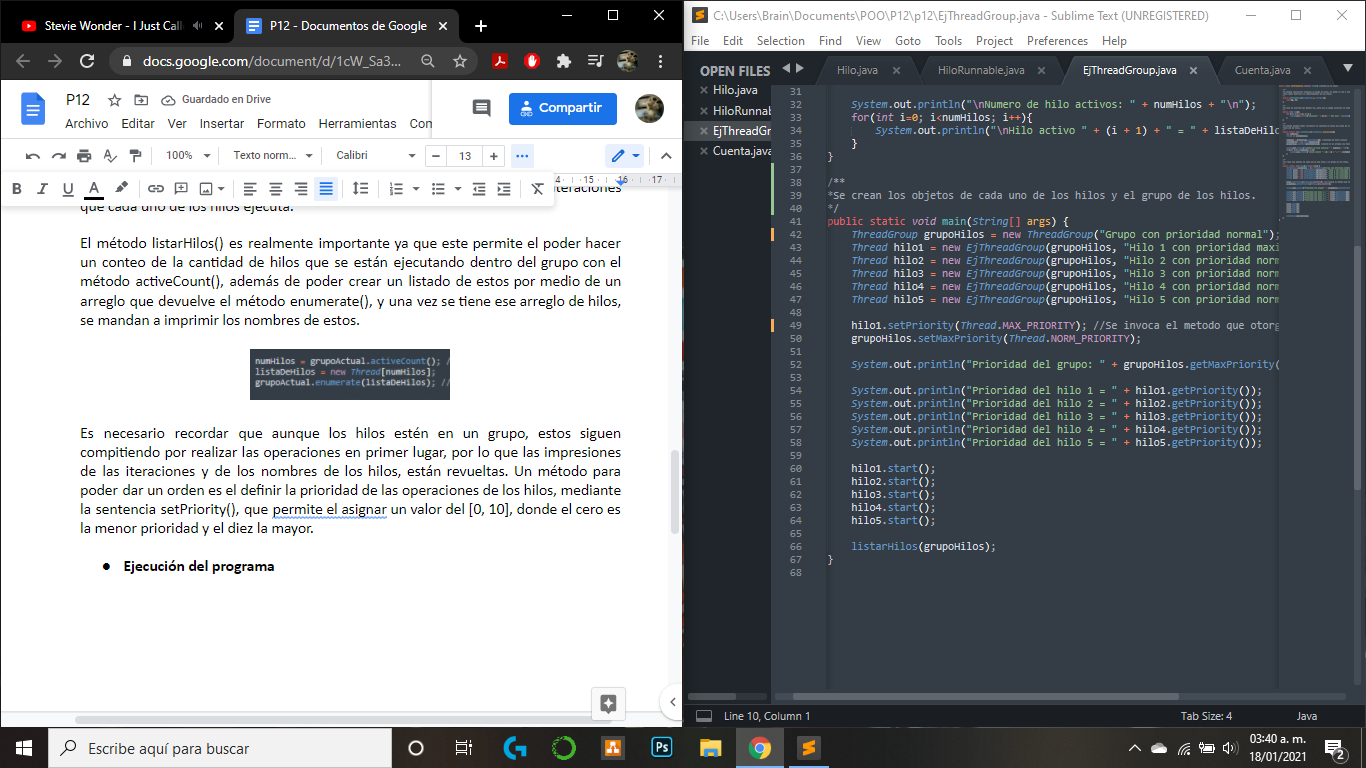


El método que se utiliza para poder comprobar la ejecución de los hilos es el método run(), del cual se hace un override, que solo ejecuta un for que indica las iteraciones que cada uno de los hilos ejecuta.

El método listarHilos() es realmente importante ya que este permite el poder hacer un conteo de la cantidad de hilos que se están ejecutando dentro del grupo con el método activeCount(), además de poder crear un listado de estos por medio de un arreglo que devuelve el método enumerate(), y una vez se tiene ese arreglo de hilos, se mandan a imprimir los nombres de estos.

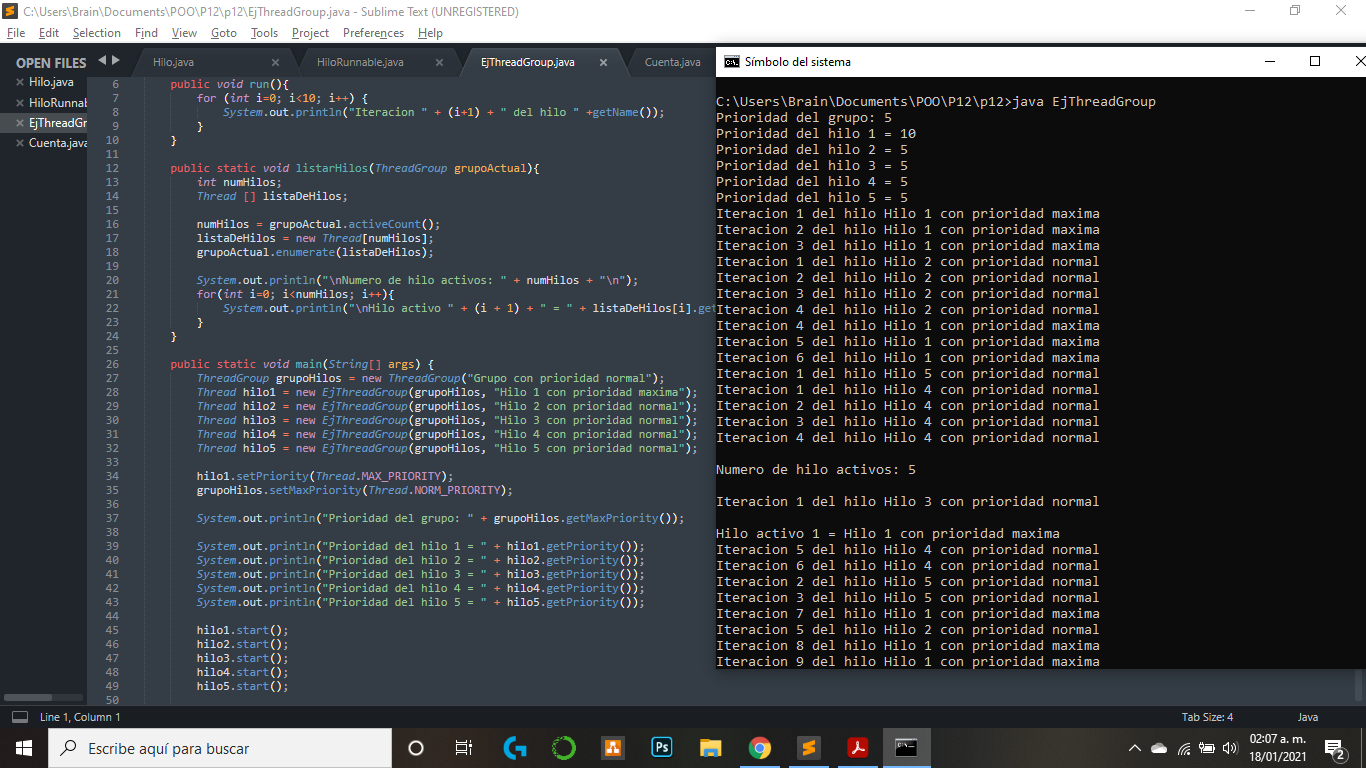


Es necesario recordar que aunque los hilos estén en un grupo, estos siguen compitiendo por realizar las operaciones en primer lugar, por lo que las impresiones de las iteraciones y de los nombres de los hilos, están revueltas. Un método para poder dar un orden es el definir la prioridad de las operaciones de los hilos, mediante la sentencia setPriority(), que permite asignar un valor del [0, 10], donde el cero es la menor prioridad y el diez la mayor.



De esta forma es posible dar un orden y poder utilizar los grupos de hilos de una forma más organizada.

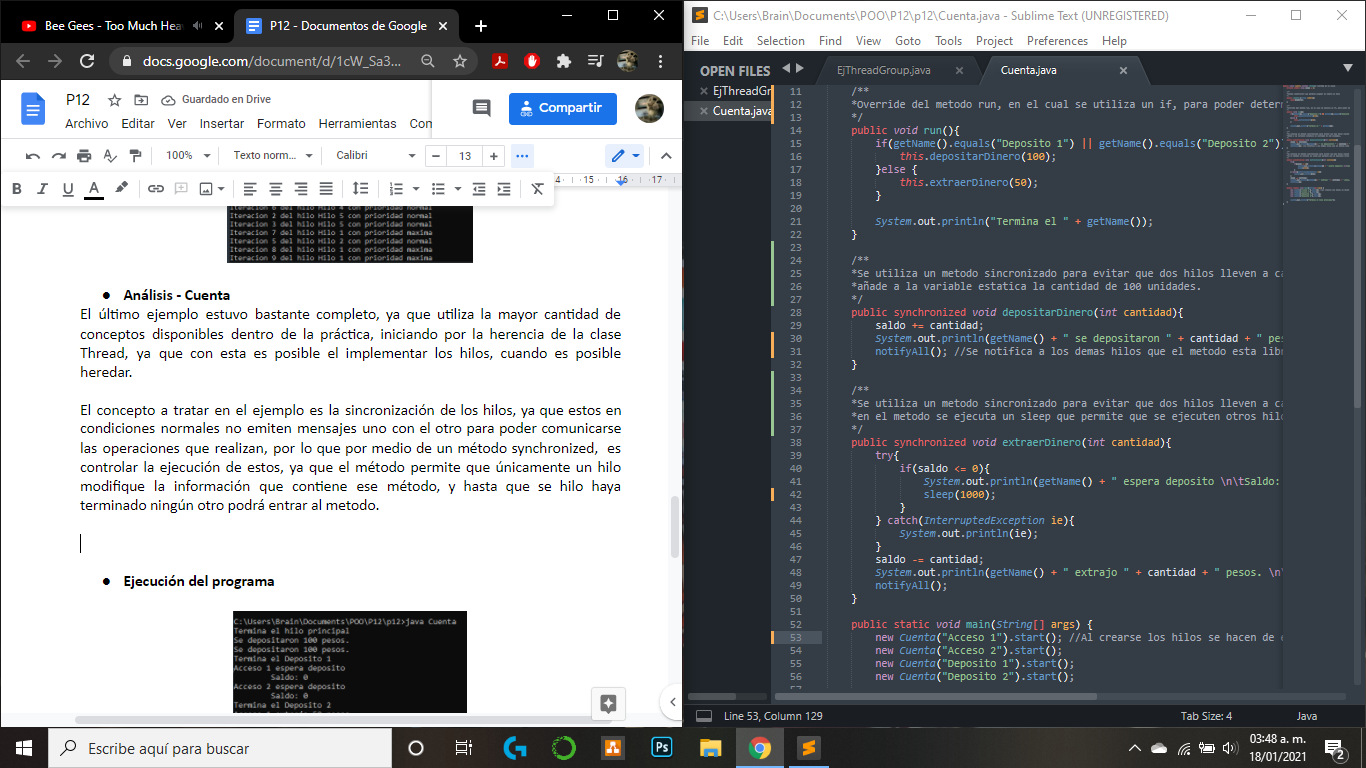
* **Ejecución del programa**

****

* **Análisis - Cuenta**

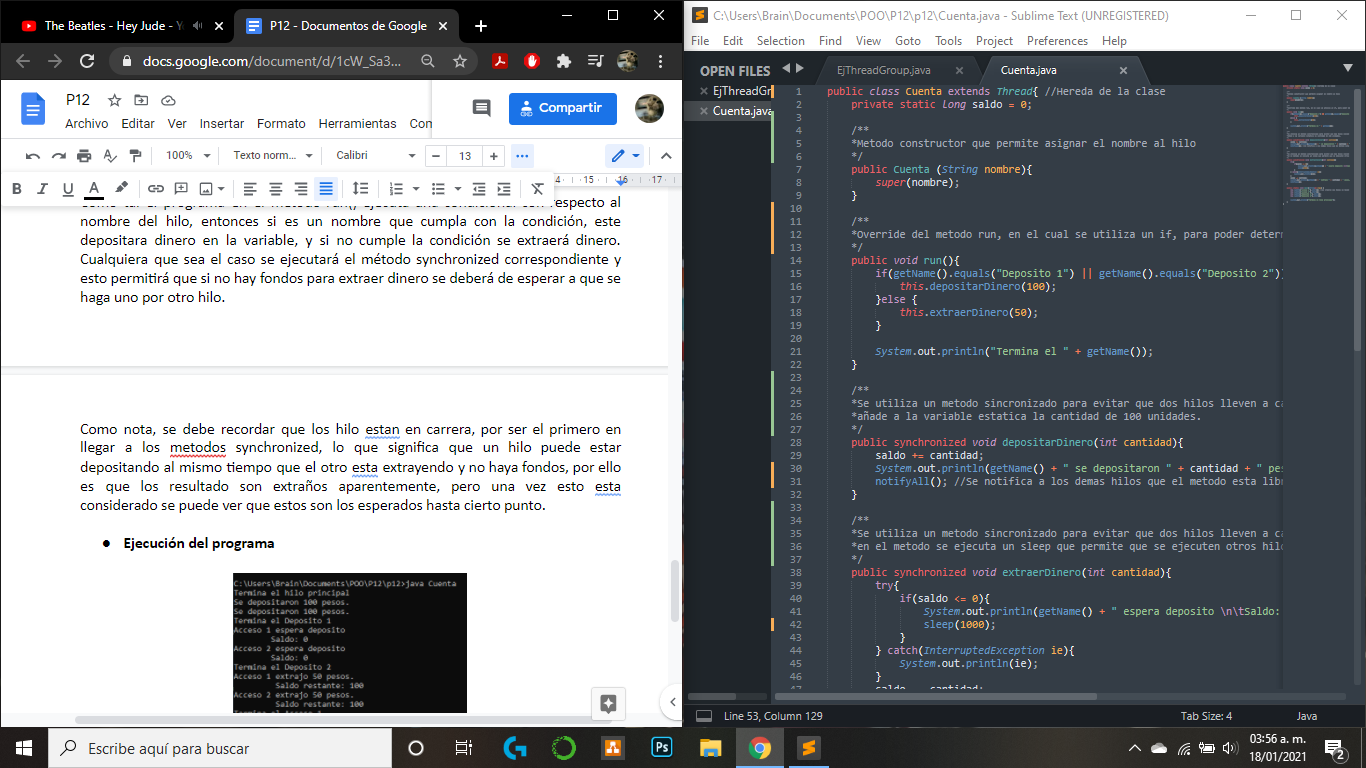
El último ejemplo estuvo bastante completo, ya que utiliza la mayor cantidad de conceptos disponibles dentro de la práctica, iniciando por la herencia de la clase Thread, ya que con esta es posible el implementar los hilos, cuando es posible heredar.

El concepto para tratar en el ejemplo es la sincronización de los hilos, ya que estos en condiciones normales no emiten mensajes uno con el otro para poder comunicarse las operaciones que realizan, por lo que por medio de un método synchronized, es posible controlar la ejecución de estos, ya que el método permite que únicamente un hilo modifique la información que contiene ese método, y hasta que se hilo haya terminado ningún otro podrá entrar al método.



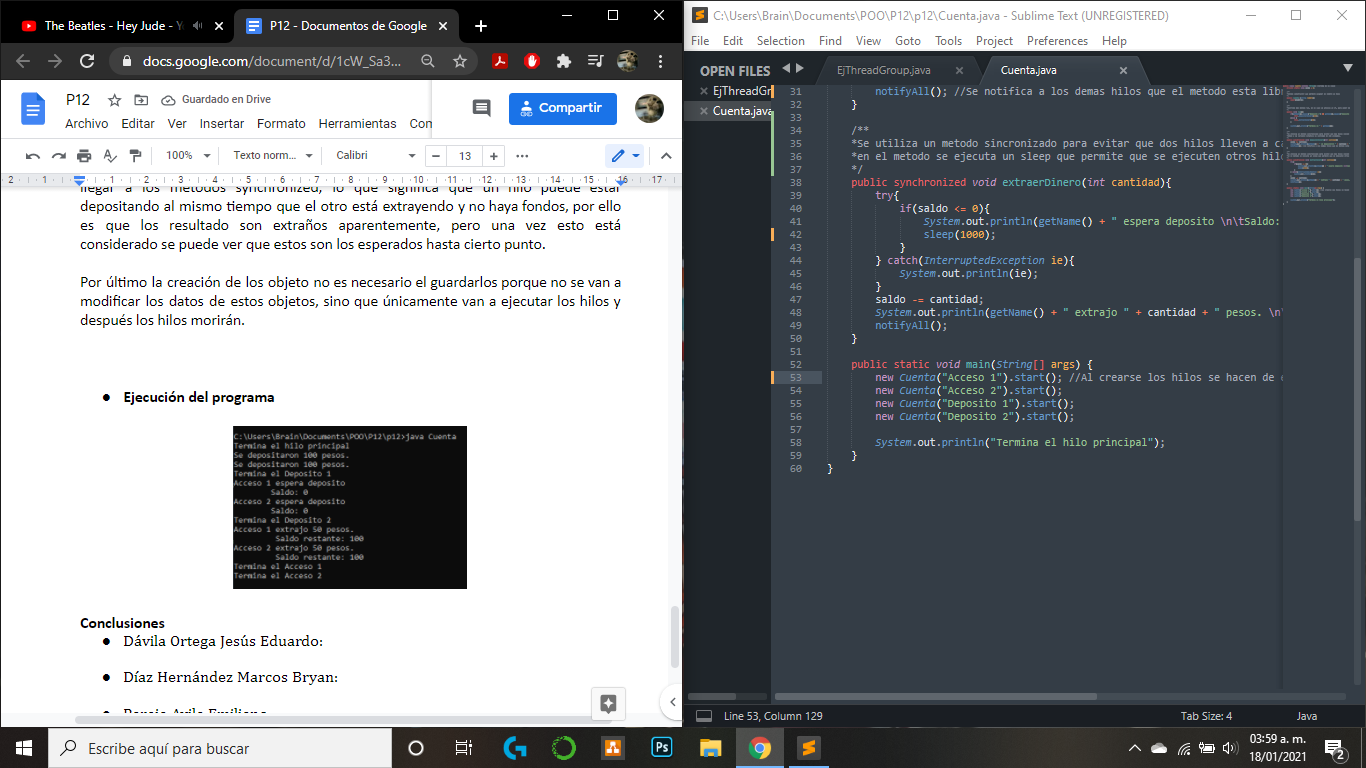
Para notificar a los demás hilos que ya ha terminado se utiliza el método notifyAll(), que permite que otro hilo acceda al método, de esta manera se puede realizar un control sobre qué información se puede modificar por múltiples hilos y cual no.

Como tal el programa en el método run() ejecuta una condicional con respecto al nombre del hilo, entonces si es un nombre que cumpla con la condición, este depositara dinero en la variable, y si no cumple la condición se extraerá dinero. Cualquiera que sea el caso se ejecutará el método synchronized correspondiente y esto permitirá que si no hay fondos para extraer dinero se deberá de esperar a que se haga uno por otro hilo.

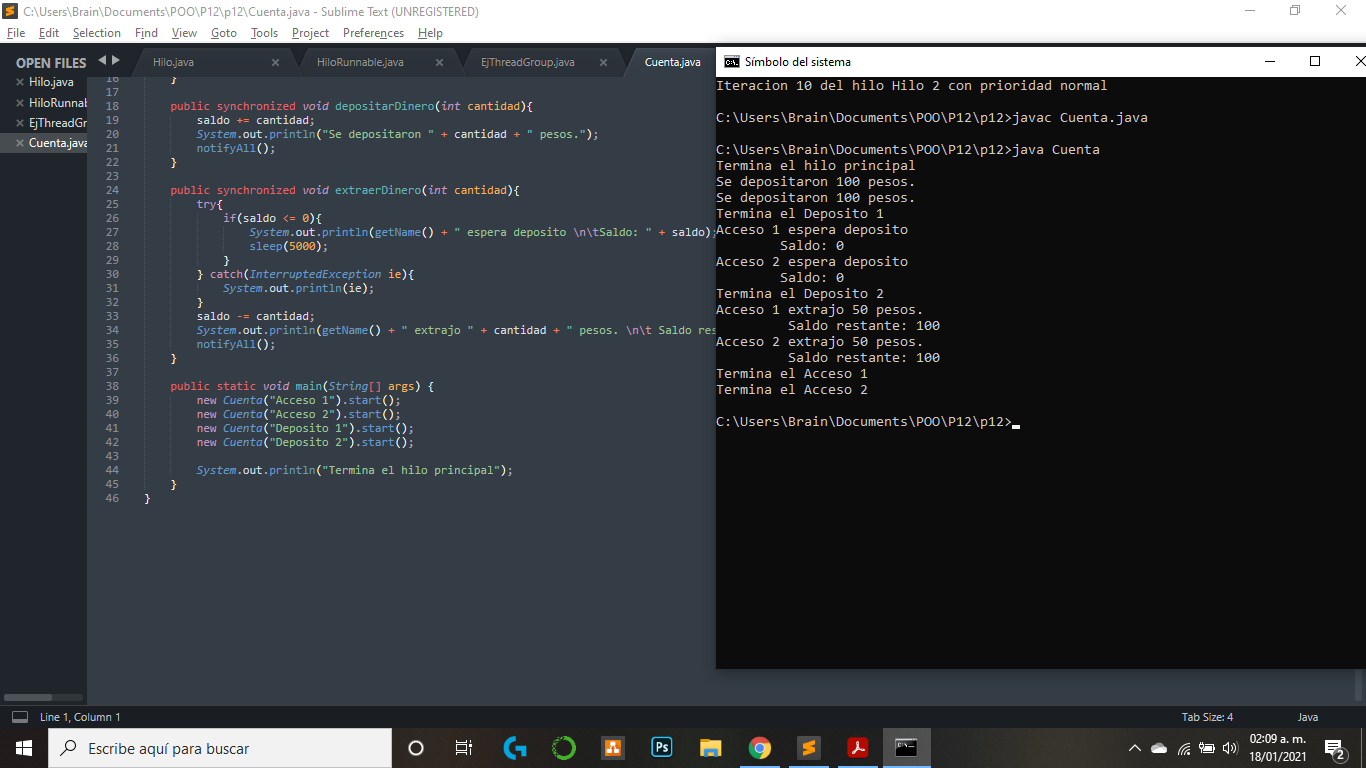


Como nota, se debe recordar que los hilo están en carrera, por ser el primero en llegar a los métodos synchronized, lo que significa que un hilo puede estar depositando al mismo tiempo que el otro está extrayendo y no haya fondos, por ello es que los resultados son extraños aparentemente, pero una vez esto está considerado se puede ver que estos son los esperados hasta cierto punto.

Por último la creación de los objetos no es necesario el guardarlos porque no se van a modificar los datos de estos objetos, sino que únicamente van a ejecutar los hilos y después los hilos morirán.



* **Ejecución del programa**

****

**Conclusiones**

* Dávila Ortega Jesús Eduardo:

Durante la práctica logramos implementar los hilos de una manera sencilla en programas los cuales se pudiera observar la diferencia al momento de implementar hilos, y gracias al complemento visto durante EDA II la comprensión del funcionamiento del programa se hizo de una manera más sencilla y así mismo se puede identificar cómo funcionan los hilos y los distintos tipos de usos que estos pueden tener en programas a futuro o incluso en programas utilizados hoy en día.

* Díaz Hernández Marcos Bryan:

Al revisar los ejemplos y en complemento con la materia de EDA II, fue más sencillo el comprender los conceptos que se manejan en la práctica, de forma que al analizar cada uno de los ejemplos podía notar como los hilos se ejecutaban e interactuaban con las variables e instrucciones, por lo que considero que se cumplió el objetivo ya que pude reconocer cómo se implementan los hilos.

* Pareja Ávila Emiliano:

Pudimos implementar hilos en programas sencillos, pero que ayudan a entender perfectamente la abstracción de un hilo dentro de un lenguaje de programación orientado a objetos, en este caso el concepto de multitareas se ve en la creación de varios hilos que realizarán una misma tarea que en específico a esta práctica, eran las instrucciones definidas dentro del método run en los diferentes ejercicios. También pude aprender que es más práctico que nuestras clases creadas hereden de la clase Thread, en lugar de implementar la interfaz Runnable, aunque también ya conozco cómo utilizar estas dos formas para crear hilos. Por lo tanto los objetivos de esta práctica se cumplieron.

* Vázquez Zavala Oliver Alexis:

Esta práctica me ha dejado de aprendizaje la importancia de los hilos en la programación, ya que la gran mayoría de aplicaciones actuales implementan hilos para poder llevar acabo sus diversas tareas, además pude conocer la forma en que son implementados los hilos en Java, mediante la herencia de la clase Thread o implementando la interfaz Runnable y conocer además las diversas funcionalidades que se tienen al crear un hilo y el ciclo de vida que siguen, aspectos que son de gran importancia considerar al momento de elaborar un proyecto.

**Bibliografía:**

* Universidad de Alicante. (2012). *Hilos*. 16/01/2021. Universidad de Alicante. Recuperado de: <http://www.jtech.ua.es/dadm/restringido/java/sesion05-apuntes.html#:~:text=En%20Java%20los%20hilos%20est%C3%A1n,definir%20el%20m%C3%A9todo%20run()>.
* Laura S. (s.f). *Hilos Java.* 16/01/2021. FI UNAM. Recuperado de: http://lcomp89.fi-b.unam.mx/licad/assets/ProgramacionMultitareasHilos/HILOS\_\_JAVA.pdf