UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Reporte Práctica 1

ALUMNO

López Diego Gabriela - 318243485

PROFESOR

Gilde Valeria Rodríguez Jiménez

AYUDANTES

Luis Angel Leyva Castillo
Rogelio Alcantar Arenas
Gibrán Aguilar Zuñiga

ASIGNATURA

Cómputo Concurrente

12 de febrero de 2024

1. Cuestionario

1. ¿Por que se utiliza Interrupted Exception en el método Main?
   Como utilizamos el metodo Thread.sleep(milisegundos), es posible que el hilo se vea interrupido por otro mientras *"duerme"* y obtengamos esta excepción. Manejamos esta excepción usando try−catch en el metodo run.

2. ¿Para que sirve el método Join?
   Suele utilizarse para otorgarle un orden a la secuencia de hilos que estemos utilizando. Se detiene el hilo principal hasta que los hilos que hemos creado completen su ejecución.

3. ¿Qué pasa si no le hacemos Join a los hilos?
   Si no mandamos a llamar el método join en los hilos secundarios, podriamos ocasionar que el hilo principal termine antes que ellos. Esto generaría resultados no deseados y no guardariamos cambios significativos y de los cuales podría depender el hilo principal.

4. ¿Cuáles son ventajas de implementar Runnable contra extender de Thread?

   - La clase que implementa Runnable puede implementar de otras interfaces
   - La clase que implementa Runnable puede además, heredar de otra clase
   - Flexibilidad de código

5. ¿Cuál es la diferencia de implementar Runnable contra Callable?
   La principal diferencia es que en Callable, se devuelve un tipo o resultado génerico y propaga excepciones. Además, Runnable suele ser más fácil de implementar y utilizar.

6. ¿Se puede predecir el orden en el que se imprime el mensaje de la clase Hilos?
   No, el resultado puede ir variando en cada ejecución. Esto se debe a que los hilos se ejecutan de manera simultanea, es decir, un hilo no necesariamente va a esperar a que el hilo anterior a el termine su ejecucion para poder realizar su respectiva tarea asignada.

7. En el archivo Hilos2.java ¿Qúe pasa si sacamos la instancia de la clase "h"de t1, es decir poner h antes de declarar t1?
   Esencialmente, no hay ningún cambio significativo. Al crear una instancia de Hilos antes de t1 asegura que solo utilizaremos esa instancia en lugar de crear una cada vez que se llame a t1.

8. Escribe que variables son locales (variables que estan en memoria del hilo) y que variables son compartidas de cada archivo y el por qué. Puedes tomarle captura al código y encerrar en un recuadro dichas variables.

   - contador/Contador.java
     - Variables locales:i
     - Variables compartidas: valor, RONDAS

     La variable i es local ya que se crea para cada hilo que entra en el for en el método suma y valor es compartido ya que los hilos h1 y h2 acceden a ella y la modifican. También cada hilo tiene acceso a la variable RONDAS.
   - contador/ContadorC.java
     - Variables locales: i, total
     - Variables compartidas: contador, RONDAS, HILOS

     En este caso, las variables locales son i y total. Ya que la variable i se crea cada que se entra en el for para cada hilo de forma independiente y total es la suma total acumuladada por el bucle for. Por otro lado, la variable compartida es contador ya que los hilos pueden acceder a ella de manera simultanea y pueden modificarla. También los hilos acceden y leen las variables RONDAS E HILOS para poder modificar contador.
   - contador/ContadorS.java
     - Variables locales: i

- Variables compartidas: valor, RONDAS

La variable i es local ya que se crea para cada hilo que entra en el for en el método suma y valor es compartido ya que los hilos h1 y h2 acceden a ella y la modifican. También cada hilo tiene acceso a la variable RONDAS.

- herencia/Hilos.java
  - Variables locales: i, nombre
  - Variables compartidas: valor, RONDAS

  La variable i y nombre son locales ya que i se crea para cada hilo que entra en el for en el método suma y nombre es asignado a cada hilo. Por otro lado, valor es compartido ya que los hilos contador1,contador2 y contador 3 acceden a ella y la modifican. También cada hilo tiene acceso a la variable RONDAS.
- hilos/Hilos.java
  - Variables locales: a,b,ID
  - Variables compartidas: Ninguna

  Las variables a,b e ID son locales ya que son creadas para cada hilo. En este caso, no tenemos alguna variable compartida. No existe variable que sea accesible y modificable por los hilos de manera simultanea.
- hilos/Hilos2.java
  - Variables locales: Ninguna
  - Variables compartidas: Ninguna

  En este archivo no tenemos variables de algún tipo, ni locales ni compartidas.

9. Contesta las preguntas de la sección de Synchronized
   Este método te grantiza que no habŕa 2 hilos (o más) ejecutando un mismo bloque de código al mismo tiempo, ya que se estarán *"turnando"* esperando a cualquier otro hilo termine su ejecución. De esta manera, evitamos problemas como condiciones de carrera y conseguiremos los resultados esperados.
   En el archivo CondicionS.java al descomentar el método **suma** pero que utiliza **synchronized** vemos que ahora el resultado de valor es el esperado y siempre consistente, independientemente de cuantas veces ejecutemos el programa pues siempre devuelve valor = 20,000.

10. ¿Como podriamos darle un comportamiento diferente a los hilos?
    Podríamos crear varias clases que extiendan de Thread o extiendan de Runnable y definir el metodo Run diferente para cada uno de ellos.

11. Escribe lo aprendido sobre esta práctica, así como tus conclusiones. También comparte si tuviste dificultades y los descubrimiento o alguna cosa de interés.
    Me agrado bastante la práctica. En código venian varios ejemplos y explicaciones que facilitarón mucho el entender el tema. He aprendido la utilidad y objetivo principal de usar Thread.sleep() al igual que Synchronized(). AL principio me tomo un poco de tiempo el ver la ejecución de varios hilos de manera simultánea, y por ende, como funcionaba el acceder y modificar una variable compartida sin condiciones de carrera. Sin embargo, con los ejercicios ha quedado más claro. Con lo que llegue a tener un poco de dificultades fue sobre los requerimientos y especificaciones de la práctica.

2. Bibliografía

- (S/f). Baeldung.com. Recuperado el 12 de febrero de 2024, de https://www.baeldung.com/java-interrupted-exception
- Sincronización de hilos. (2009, junio 13). Recuperado el 12 de febrero de 2024, de https://eaddfsi.wordpress.com/2009/06/13/sincronizacion-de-hilos/
- Baeldung (2024, 8 enero). Runnable vs. Callable in Java . Recuperado el 12 de febrero de 2024, de https://www.baeldung.com/java-runnable-callable

- La clase derivada de Thread. (s.f.). Recuperado el 12 de febrero de 2024, de http://www.sc.ehu.es/sbweb/fisica/cursoJava/applets/threads/subprocesos.htm
- Metodo synchronized de forma correcta. (s/f). Stack Overflow en español. Recuperado el 12 de febrero de 2024, de https://es.stackoverflow.com/questions/265850/como-se-usa-el-metodo-synchronized-de-forma-correcta

3. SonarLint

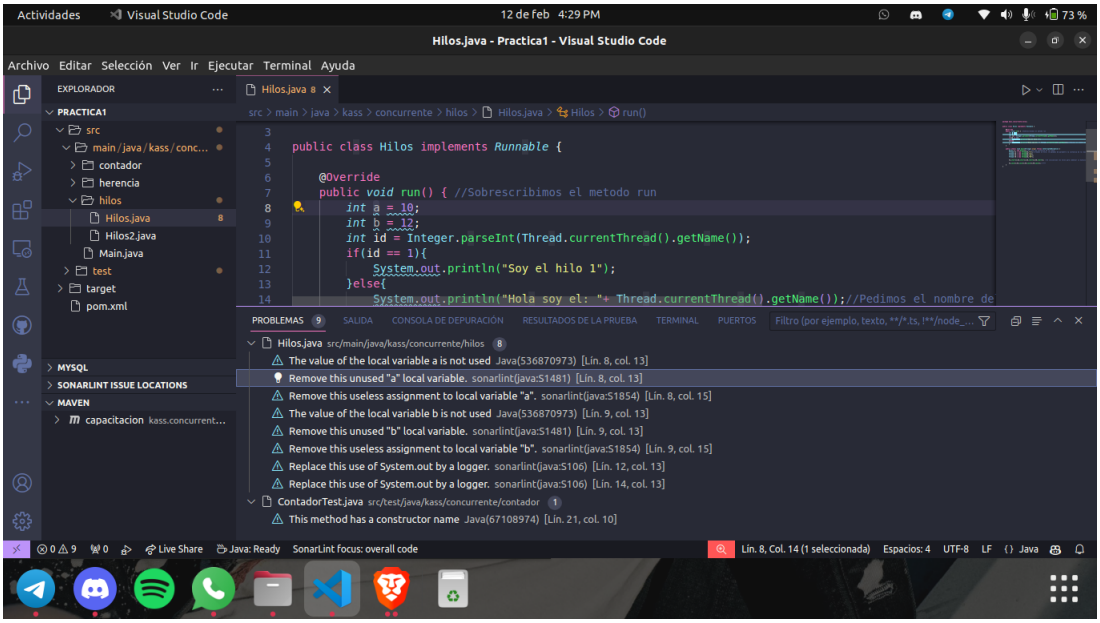

Figura 1: Clase hilos/Hilos.java

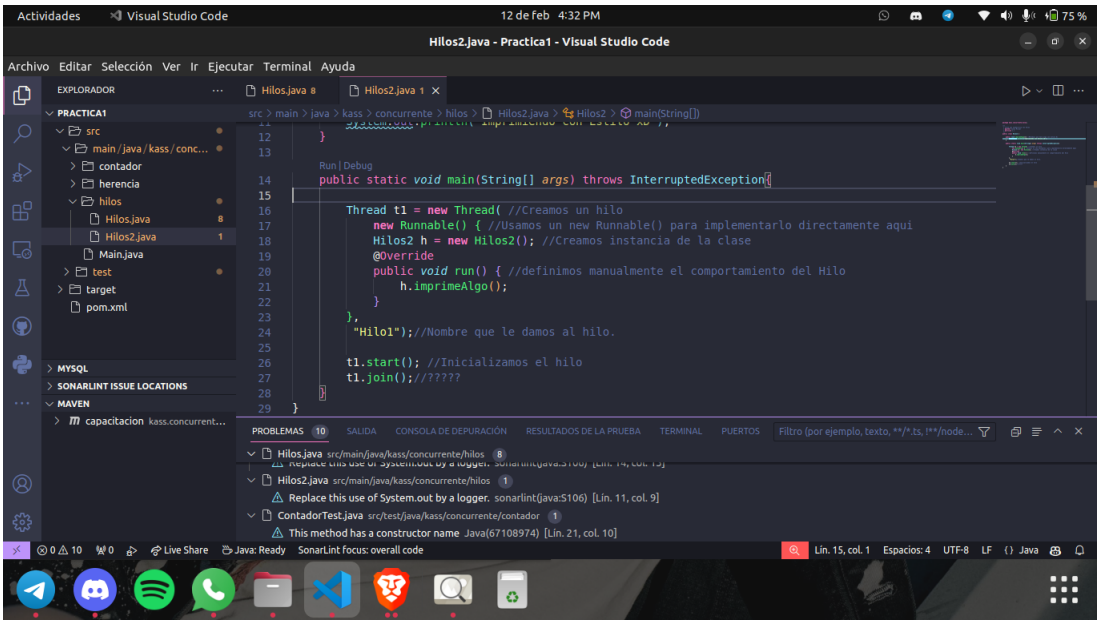

Figura 2: Clase hilos/Hilos2.java

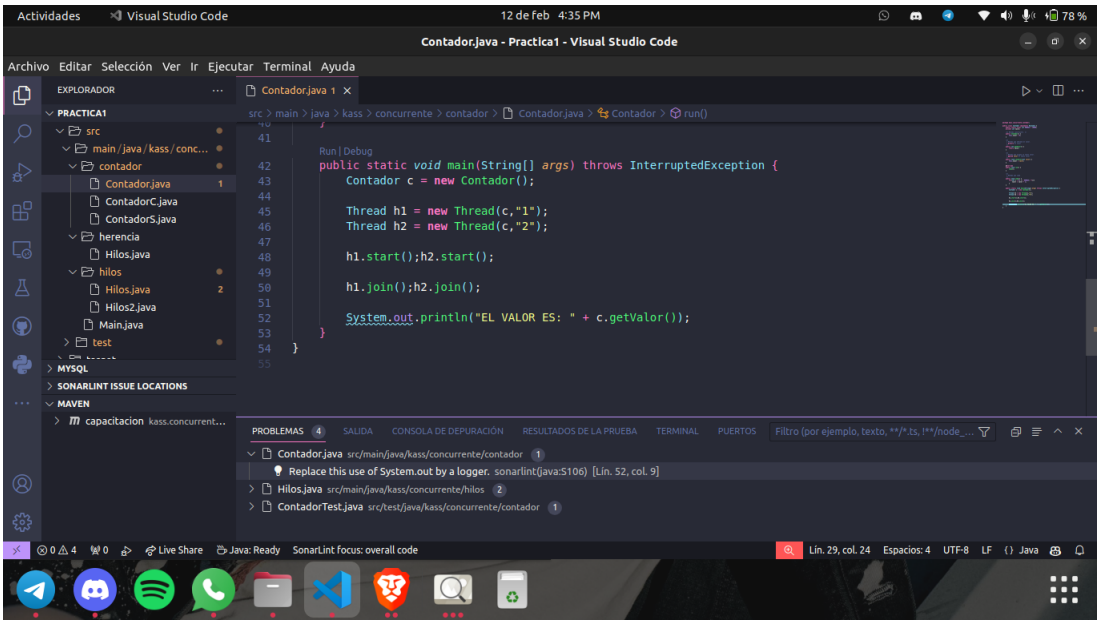

Figura 3: Clase contador/Contador.java

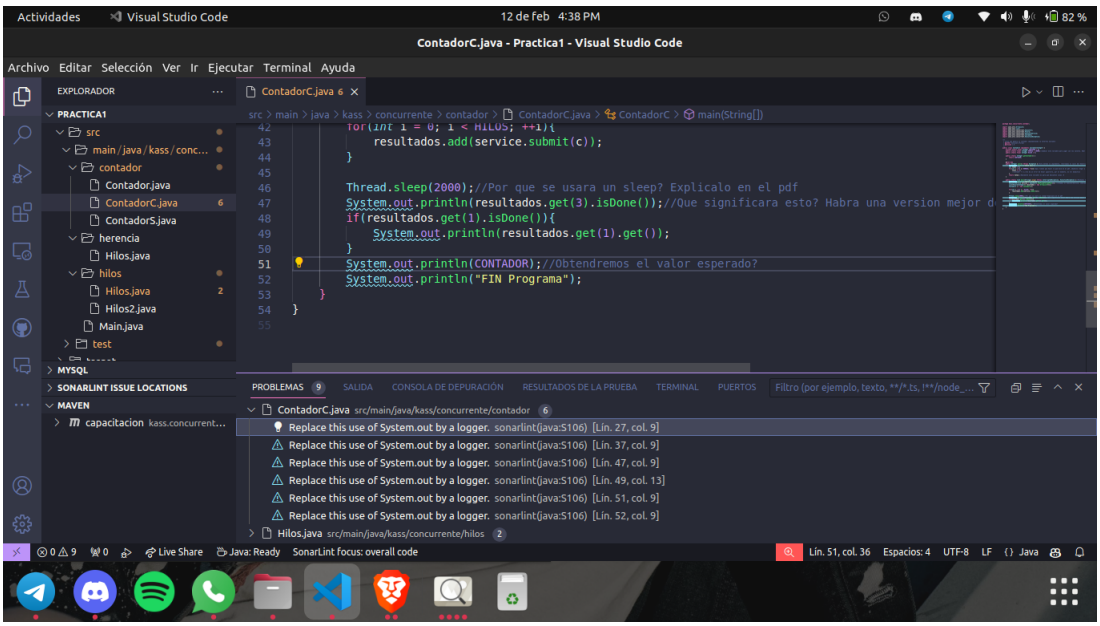

Figura 4: Clase contador/ContadorC.java

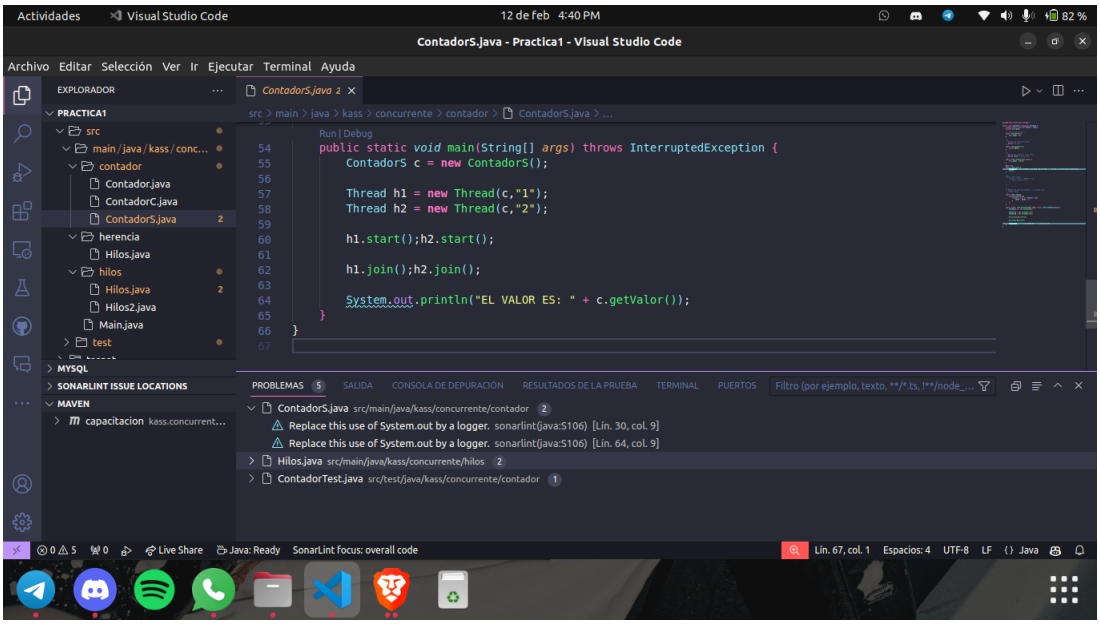

Figura 5: Clase contador/ContadorS.java

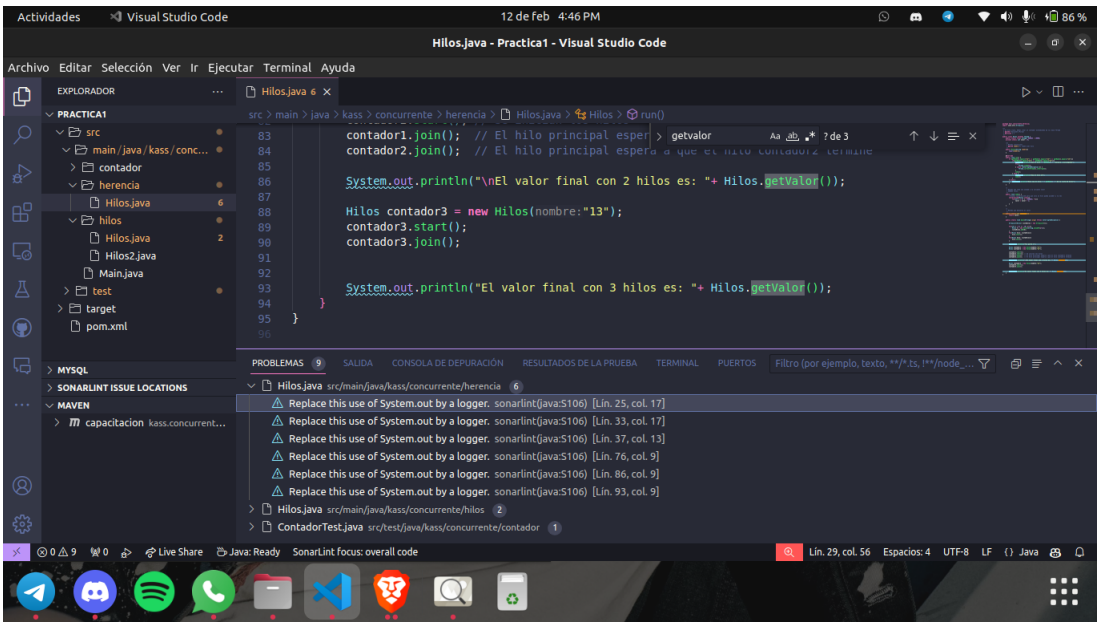

Figura 6: Clase herencia/Hilos.java

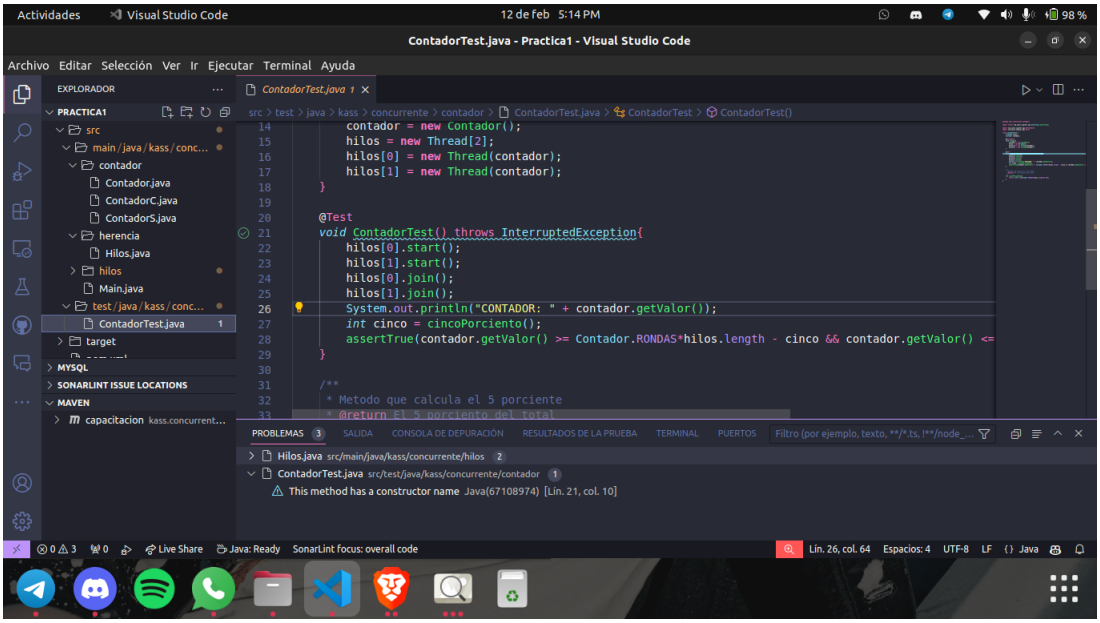

Figura 7: Clase test/ContadorTest.java