Brian Camilo Fajardo Sánchez

**Recuperación Parcial Corte 1**

Para la explicación, se van a detallar paso a paso como se solucionó cada punto. Tener en cuenta que a medida que avanza el parcial se modificaron métodos y los atributos. Por lo que si desde el punto uno se desea validar las instrucciones con el código no se verá acorde a lo explicado ya que en el repositorio se encuentra la versión final.

1. Cree una clase de tipo Thread que represente el ciclo de vida de un hilo que calcule una parte de los dígitos requeridos.

* Para solucionar este punto, inicialmente se creó una clase llamada *PiDigitsThread.java,* que heredaba de la clase *Thread* y se agregó toda la lógica referente al calculo de la fórmula *BBP.*
* En la nueva clase, en vez de tener un método llamado *getDigits*, hay un método llamado *calculate* que ejecuta la misma función.
* Se añadió un atributo de clase llamado *answer* que es un arreglo de bytes destinado a almacenar el resultado del cálculo de la función *calculate*
* Como *PiDigitsThread* es un thread, se sobrescribió el método *run* y en su contenido solo tiene el llamado al método *calculate* que se almacenará en la variable *answer,* el resto de los métodos se dejaron igual*.*
* En la clase *PiDigits* se creó un nuevo método para reemplazar el antiguo getDigits, que inicialmente (antes de la implementación del punto 2 y 3) únicamente contenía un llamado a la clase *PiDigitsThread* y funcionaba como si contara con un solo hilo en la ejecución.

1. Haga que la función *PiDigits.getDigits()* reciba como parámetro adicional un valor N, correspondiente al número de hilos entre los que se va a paralelizar la solución. Haga que dicha función espere hasta que los N hilos terminen de resolver el problema para combinar las respuestas y entonces retornar el resultado. Para esto, puede utilizar el método *Join()* del API de concurrencia de Java.

* Para este punto, primero se añadió el parámetro N al método *getDigits* de la clase *PiDigits*. Se crearon múltiples métodos que se explican a continuación y se realizaron sus respectivas llamadas desde el *getDigits*.
* Se creó un arreglo de bytes llamado *answer* que almacenará el resultado de los threads y un arreglo de *PiDigitsThread* que almacenará los threads
* Se creó un método llamado *createThreads* el cual le entra como atributo el arreglo de los threads, el punto de inicio general (es decir desde donde se van a iniciar a calcular los núemros), la cantidad de números que se desea calcular y la cantidad de hilos. Este método inicialmente calcula el módulo de la cantidad de número que se desea calcular y el número de hilos que se especificaron, esto con el fin de saber cuántos números sobran al hacer la división entera y poder asignarlos más adelante. Luego se calcula el rango inicial que será la división entera de la cantidad de números a calcular entre la cantidad de hilos. Se asigna el valor de inicio y se crea un bucle for que iterará por el número de hilos. Seguido se realiza una verificación para saber si ya se va a calcular el rango de último hilo, si sí, al rango se le añade el residuo de la división que almacenamos al inicio del método, si no, continua y agrega un nuevo thread al arreglo de hilos que entró por parámetro. Por último, se le agrega el rango del último hilo al punto de inicio del siguiente hilo. Al salir del bucle, se hace otro bucle inicializando todos los hilos.
* Se creó un método llamado *joinThreads* al cual le entran únicamente el arreglo de hilos y su función es recorrer el arreglo y realizar un *join()* para que todos los hilos termines su ejecución antes de que intente obtener el resultado.
* Para finalizar, se creó un método llamado *getAnswer* que le entra por parámetro el arreglo de hilos y el arreglo vacío de donde se almacenarán las respuestas. Para obtener las respuestas, en la clase *PiDigitsThread* se creó un método llamado *getAnswer.* En este método se hace uso de un bucle para recorrer los hilos, obtener sus respuestas y copiarlas al arreglo *answer* de la clase *PiDigits*, en cada iteración se calcula el nuevo punto de partida del siguiente hilo.

1. Ajuste la implementación para que cada 5 segundos los hilos se detengan e impriman el número de dígitos que han procesado y una vez se presione la tecla enter que los hilos continúen su proceso.

* Para solucionar este punto, se inició modificando el método *getDigits* de la clase *PiDigits* donde se añadió la creación de dos variables, un AtomicInteger *counter*, que servirá para contar la cantidad de dígitos que han procesados y una variable Object *block* que funcionará para controlar los hilos (wait, synchronized, notifyAll…). Luego, se quitó el método *joinThreads* para agregar uno nuevo llamado *waitThreads* que se explicará más adelante.
* Al método *createThreads* se le añadieron las dos nuevas variables que se crearon en el paso anterior y se le enviaron por parámetro a los hilos. Su uso se explica más adelante.
* En la clase *PiDigitsThread* se realizaron varias modificaciones, primero, se añadieron 2 variables de clase nuevos, un AtomicInteger *counter*, que contará la cantidad de dígitos que han procesados y una variable Object *block,* que funcionará para controlar los hilos (wait, synchronized, notifyAll…). Estos entran por el constructor.
* Se creó un método llamado *pauseThread* que cumple la función de realizar un *wait* del hilo actual haciendo uso del *lock* en una sección síncrona.
* Dentro del método mencionado en el paso 1, *calculate*, se modificó la última parte para que tuviera en cuenta el tiempo que lleva el hilo ejecutándose y que haga el llamado al método del paso anterior a los 5 segundos. Además, hay región crítica para incrementar el valor del contador tipo atómico, por cada dígito calculado.
* Volviendo a la clase *PiDigits* el método quizá más importante de este punto se llama *waitThreads.* El cual se llama desde el método *getDigits* justo después de la creación de los hilos. El método *waitThreads,* lo que realiza es el control de los tiempos de ejecución, es decir mide cuanto tiempo lleva él ejecutándose, para que a los 5 segundos el usuario pueda ver el número de dígitos que han procesado y que reciba como una entrada de usuario el enter requerido y reactive nuevamente los hilos hijos que contiene. Para contar el tiempo se calcula la diferencia de tiempos entre el tiempo en milisegundos antes de iniciar el bucle y luego en cada iteración de este. Para imprimir el valor de los dígitos que han procesados, se hace un *get* al AtomicInteger y se imprime su valor, esto funciona ya que es un dato atómico y controla la concurrencia entre hilos. Para reactivar los hilos, se crea una región crítica en el que se hace un *notifyAll* por medio del objeto lock (el cual lo tienen todos los hilos hijos) y por último, para salir del buble y validar la respuesta de los n dígitos de se hace una validación en cada iteración del bucle en donde se revisa si los hilos siguen vivos, si sí sale de bucle, termina el método e imprime el resultado al llamar a *getAnswer*, si no, sigue el bucle infinito.