

En esta tarea vamos a desarrollar un programa distribuido, el cual calculará una aproximación de PI utilizando la serie de [Gregory-Leibniz](https://crypto.stanford.edu/pbc/notes/pi/glseries.html).

La serie tiene la siguiente forma:  4/1-4/3+4/5-4/7+4/9-4/11+4/13-4/15 ...

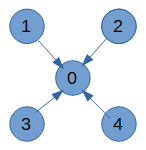
Notar que los denominadores son los números impares:  1, 3, 5, 7, 9, 11, 13 ...

El programa va a ejecutar en forma distribuida sobre cinco nodos, cada nodo sería una computadora diferente.

Por lo pronto, vamos a probar el programa en una sola computadora utilizando cinco ventanas de comandos de Windows o cinco terminales de Linux, en cada ventana representará un nodo (una instancia del programa).

Cada nodo (excepto el nodo 0) deberá calcular 10 millones de términos de la serie.

Implementaremos la siguiente topología lógica de tipo estrella, cada nodo se ha identificado con un número entero:



El nodo 0 actuará como servidor y los nodos 1, 2, 3 y 4 actuarán como clientes.

Se debe desarrollar un solo programa, por tanto  será necesario pasar como parámetro al programa el número de nodo actual, de manera que el programa pueda actual como servidor o como cliente, según el número de nodo que pasa como parámetro.

Consideremos el siguiente programa:

class PI  
{

  static Object lock = new Object();  
  static double pi = 0;

  static class Worker extends Thread

  {  
    Socket conexion;  
    Worker(Socket conexion)  
    {  
      this.conexion = conexion;  
    }  
    public void run()  
    {

      // **Algoritmo 1**

    }  
  }  
  public static void main(String[] args) throws Exception  
  {  
    if (args.length != 1)  
    {  
      System.err.println("Uso:");

      System.err.println("java PI <nodo>");  
      System.exit(0);  
    }  
    int nodo = Integer.valueOf(args[0]);  
    if (nodo == 0)  
    {

      // **Algoritmo 2**  
    }  
    else  
    {

      // **Algoritmo 3**  
    }  
  }  
}

Se propone implementar los siguientes algoritmos:

**Algoritmo 1**

1. Crear los streams de entrada y salida.

2. Declarar la variable "x" de tipo double.  
3. Recibir en la variable "x" la suma calculada por el cliente.

4. En un bloque synchronized mediante el objeto "lock":

4.1 Asignar a la variable "pi" la expresión: x+pi

5. Cerrar los streams de entrada y salida.  
6. Cerrar la conexión "conexion".

**Algoritmo 2**

1. Declarar una variable "servidor" de tipo ServerSocket.  
2. Crear un socket servidor utilizando el puerto 50000 y asignarlo a la variable "servidor".  
3. Declarar un vector "w" de tipo Worker con 4 elementos.  
4. Declarar una variable entera "i" y asignarle cero.  
5. En un ciclo:

5.1 Si la variable "i" es igual a 4, entonces salir del ciclo.  
5.2 Declarar una variable "conexion" de tipo Socket.  
5.3 Invocar el método servidor.accept() y asignar el resultado a la variable "conexion".  
5.4 Crear una instancia de la clase Worker, pasando como parámetro la variable "conexion". Asignar la instancia al elemento w[i].  
5.5 Invocar el método w[i].start()  
5.6 Incrementar la variable "i".  
5.7 Ir al paso 5.1

6. Declarar una variable "i" entera y asignarle cero.

7. En un ciclo:

7.1 Si la variable "i" es igual a 4, entonces salir del ciclo.

7.2 Invocar el método w[i].join()

7.3 Incrementar la variable "i".

7.4 Ir al paso 7.1

8. Desplegar el valor de la variable "pi".

**Algoritmo 3**

1. Declarar la variable "conexion" de tipo Socket y asignarle null.

2. Realizar la conexión con el servidor implementando re-intento. Asignar el socket a la variable "conexion".

3. Crear los streams de entrada y salida.

4. Declarar la variable "suma" de tipo double y asignarle cero.

5. Declarar una variable "i" de tipo entero y asignarle cero.

6. En un ciclo:

6.1 Si la variable "i" es igual a 10000000, entonces salir del ciclo.

6.2 Asignar a la variable "suma" la expresión:  4.0/(8\*i+2\*(nodo-2)+3)+suma

6.3 Incrementar la variable "i".

6.4 Ir al paso 6.1

7. Asignar a la variable "suma" la expresión:  nodo%2==0?-suma:suma

8. Enviar al servidor el valor de la variable "suma".

9. Cerrar los streams de entrada y salida.  
10. Cerrar la conexión "conexion".

Notar que el algoritmo 1 se deberá ejecutar dentro de un bloque try.

También se debe notar que la variable "lock" debe ser estática para que todos los threads accedan al mismo lock.

Se deberá subir a la plataforma un **archivo en formato ZIP** que contenga el código fuente del programa desarrollado y un reporte como **documento PDF** con la captura de pantalla de la compilación y ejecución del programa. El reporte deberá incluir portada,  una descripción de cada captura de pantalla y conclusiones. No se admitirán archivos en formato Word o en formato RAR.

Valor de la tarea: 30% (1.8 punto de la primera evaluación parcial)