Informe de Laboratorio 03

Tema: Tecnologia de Objetos - Laboratorio 03

Nota

Estudiante	Escuela	Asignatura
Chambilla Perca Ricardo Mauricio rchambillap@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Tecnologia de objetvos Semestre: VI Código:

Laboratorio	Tema	Duración
03	Tecnologia de Objetos - Laboratorio 03	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2025 - B	Del 22 de Septiembre 2025	Al 29 de Septiembre 2025

1. Objetivos

- Programar paralelamente usando treads.
- Comparar los lenguajes de programación: Java, C++ y Go.

2. Desarrollo

- Para cualquier función definida y los puntos a y b. Hallar la integral utilizando el método del trapecio por aproximación.
- Utilizar Programación OO.
- Utilizar treads para los hilos.
- Investigar el uso de Pool de Threads.

3. Equipos, materiales y temas utilizados

- Sistema Operativo Ubuntu GNU Linux o Windows
- VIM 9.0 o Visual Studio Code
- Git.
- Cuenta en GitHub con el correo institucional.

4. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/shanccom/teo.git`
- URL para el laboratorio 01 en el Repositorio GitHub.
- `https://github.com/shanccom/teo/tree/main/lab03`

5. Actividades

5.1. Lenguaje de Programacion - Java

- Uso de threads en Java para resolver el problema con el metodo del trapecio

Listing 1: Trapecio.java

```java
interface Funcion {
    double evaluar(double x);
}

class FuncionEjemplo implements Funcion {
    public double evaluar(double x) {
        return 2 * x * x + 3 * x + 0.5;
    }
}

class TrapecioHilo extends Thread {
    private double a, b;
    private int n;
    private Funcion funcion;
    private double resultado;

    public TrapecioHilo(double a, double b, int n, Funcion funcion) {
        this.a = a;
        this.b = b;
        this.n = n;
        this.funcion = funcion;
    }

    public void run() {
        double h = (b - a) / n;
        double suma = funcion.evaluar(a) + funcion.evaluar(b);
        for (int i = 1; i < n; i++) {
            double xi = a + i * h;
            suma += 2 * funcion.evaluar(xi);
        }
        resultado = (h / 2) * suma;
    }

    public double getResultado() {
        return resultado;
    }
}

public class Trapecio {
```

```
40      public static void main(String[] args) {
41          double a = 2, b = 20;
42          int subdivisiones = 1;
43          double areaAnterior = -1;
44          boolean detener = false;
45
46          Funcion funcion = new FuncionEjemplo();
47
48          while (!detener) {
49              TrapecioHilo[] hilos = new TrapecioHilo[subdivisiones];
50              double intervalo = (b - a) / subdivisiones;
51
52              for (int i = 0; i < subdivisiones; i++) {
53                  double ai = a + i * intervalo;
54                  double bi = (i == subdivisiones - 1) ? b : ai + intervalo;
55                  hilos[i] = new TrapecioHilo(ai, bi, 1, funcion);
56                  hilos[i].start();
57              }
58
59              double areaTotal = 0.0;
60              try {
61                  for (TrapecioHilo hilo : hilos) {
62                      hilo.join();
63                      areaTotal += hilo.getResultado();
64                  }
65              } catch (InterruptedException e) {
66                  e.printStackTrace();
67              }
68
69              System.out.printf("N = %d -> rea aproximada = %.6f%n", subdivisiones,
                    areaTotal);
70
71              double areaRedondeada = Math.round(areaTotal * 10000.0) / 10000.0;
72              double areaAnteriorRedondeada = Math.round(areaAnterior * 10000.0) / 10000.0;
73
74              if (areaRedondeada == areaAnteriorRedondeada) {
75                  detener = true;
76                  System.out.println("\nSe detuvo porque el rea ya no cambia en 4
                        decimales.");
77              }
78
79              areaAnterior = areaTotal;
80              subdivisiones++;
81          }
82      }
83  }
```

- Salida de Trapecio.java

```
PS C:\Users\sdhm\Desktop\teo\lab03> java -cp . Trapecio
N = 329 -> Área aproximada = 5931,017960
N = 330 -> Área aproximada = 5931,017851
N = 331 -> Área aproximada = 5931,017744
N = 332 -> Área aproximada = 5931,017637
N = 333 -> Área aproximada = 5931,017531
N = 334 -> Área aproximada = 5931,017426
N = 335 -> Área aproximada = 5931,017322
N = 336 -> Área aproximada = 5931,017219
N = 337 -> Área aproximada = 5931,017117
N = 338 -> Área aproximada = 5931,017016
N = 339 -> Área aproximada = 5931,016916
N = 340 -> Área aproximada = 5931,016817
N = 341 -> Área aproximada = 5931,016718
N = 342 -> Área aproximada = 5931,016620
N = 343 -> Área aproximada = 5931,016524
N = 344 -> Área aproximada = 5931,016428
N = 345 -> Área aproximada = 5931,016333
N = 346 -> Área aproximada = 5931,016238
N = 347 -> Área aproximada = 5931,016145
N = 348 -> Área aproximada = 5931,016052

Se detuvo porque el área ya no cambia en 4 decimales.
PS C:\Users\sdhm\Desktop\teo\lab03>
```

- Uso de Pool de threads en Java para resolver el problema con el metodo del trapecio

Listing 2: TrapecioPool.java

```java
import java.util.concurrent.*;
import java.util.*;

interface Funcion {
    double evaluar(double x);
}

class FuncionEjemplo implements Funcion {
    public double evaluar(double x) {
        return 2 * x * x + 3 * x + 0.5;
    }
}

class TareaTrapecio implements Callable<Double> {
    private double x1, x2, base;
    private Funcion funcion;

    public TareaTrapecio(double x1, double x2, double base, Funcion funcion) {
        this.x1 = x1;
        this.x2 = x2;
        this.base = base;
        this.funcion = funcion;
    }
```

```
24
25      @Override
26      public Double call() {
27          return (base / 2.0) * (funcion.evaluar(x1) + funcion.evaluar(x2));
28      }
29  }
30
31  public class TrapecioPool {
32      public static void main(String[] args) throws Exception {
33          double limiteInferior = 2, limiteSuperior = 20;
34          double areaAnterior = -1;
35          boolean detener = false;
36          int numTrapecios = 1;
37
38          int numNucleos = Runtime.getRuntime().availableProcessors();
39          Funcion funcion = new FuncionEjemplo();
40
41          while (!detener) {
42              double base = (limiteSuperior - limiteInferior) / numTrapecios;
43
44              //Creamos un pool con los nucleos que tiene el host
45              ExecutorService ejecutor = Executors.newFixedThreadPool(numNucleos);
46              List<Future<Double>> resultados = new ArrayList<>();
47
48              for (int i = 0; i < numTrapecios; i++) {
49                  double xi = limiteInferior + i * base;
50                  double xf = xi + base;
51                  resultados.add(ejecutor.submit(new TareaTrapecio(xi, xf, base, funcion)));
52              }
53
54              double area = 0;
55              for (Future<Double> futuro : resultados) {
56                  area += futuro.get();
57              }
58
59              ejecutor.shutdown();
60
61              System.out.printf("N = %d -> rea aproximada = %.6f%n", numTrapecios, area);
62
63              double areaRedondeada = Math.round(area * 10000.0) / 10000.0;
64              double areaAnteriorRedondeada = Math.round(areaAnterior * 10000.0) / 10000.0;
65
66              if (areaRedondeada == areaAnteriorRedondeada) {
67                  detener = true;
68                  System.out.println("\nSe detuvo porque el rea ya no cambia en 4
                        decimales.");
69              }
70
71              areaAnterior = area;
72              numTrapecios++;
73          }
74      }
75  }
```

- Salida de TrapecioPool.java: Cabe aclarar que no cambian las iteraciones de N, lo que cambia es la distribucion de trabajo por hilo

```
PS C:\Users\sdhm\Desktop\teo\lab03> java -cp . TrapecioPool
N = 323 -> Área aproximada = 5931,018633
N = 324 -> Área aproximada = 5931,018519
N = 325 -> Área aproximada = 5931,018405
N = 326 -> Área aproximada = 5931,018292
N = 327 -> Área aproximada = 5931,018180
N = 328 -> Área aproximada = 5931,018070
N = 329 -> Área aproximada = 5931,017960
N = 330 -> Área aproximada = 5931,017851
N = 331 -> Área aproximada = 5931,017744
N = 332 -> Área aproximada = 5931,017637
N = 333 -> Área aproximada = 5931,017531
N = 334 -> Área aproximada = 5931,017426
N = 335 -> Área aproximada = 5931,017322
N = 336 -> Área aproximada = 5931,017219
N = 337 -> Área aproximada = 5931,017117
N = 338 -> Área aproximada = 5931,017016
N = 339 -> Área aproximada = 5931,016916
N = 340 -> Área aproximada = 5931,016817
N = 341 -> Área aproximada = 5931,016718
N = 342 -> Área aproximada = 5931,016620
N = 343 -> Área aproximada = 5931,016524
N = 344 -> Área aproximada = 5931,016428
N = 345 -> Área aproximada = 5931,016333
N = 346 -> Área aproximada = 5931,016238
N = 347 -> Área aproximada = 5931,016145
N = 348 -> Área aproximada = 5931,016052

Se detuvo porque el área ya no cambia en 4 decimales.
PS C:\Users\sdhm\Desktop\teo\lab03>
```

5.2. Lenguaje de Programacion - C++

- Uso de threads en C++ para resolver el problema con el metodo del trapecio

Listing 3: trapecio.cpp

```cpp
#include <iostream>
#include <cmath>
#include <functional>
#include <thread>
#include <vector>
#include <mutex>
#include <chrono>

// Struct que representa una funcin con una variable
struct Funcion {
    std::function<double(double)> f; // Funcin matemtica f(x)
    std::string expresion;         // Representacin en string de la funcin

    // Constructor
```

```
15      Funcion(std::function<double(double)> func, std::string expr)
16          : f(func), expresion(expr) {}
17
18      // Evaluar la funcin en un punto x
19      double evaluar(double x) const {
20          return f(x);
21      }
22
23      // Mostrar informacin de la funcin
24      void mostrar() const {
25          std::cout << "Funcin: " << expresion << std::endl;
26      }
27  };
28
29  // Funcin para calcular el rea de un trapecio
30  double areaTrapecio(double lado1, double lado2, double altura) {
31    return ((lado1 + lado2) * altura) / 2.0;
32  }
33
34  // Funcin para generar puntos equidistantes entre x1 y x2
35  std::vector<double> generarPuntos(double x1, double x2, int c) {
36    std::vector<double> puntos;
37    puntos.reserve(c + 1);
38
39    double paso = (x2 - x1) / c;
40
41    for (int i = 0; i <= c; i++) {
42      puntos.push_back(x1 + i * paso);
43    }
44
45    return puntos;
46  }
47
48  const int CANTIDAD_MAXIMA_DE_HILOS = 20;
49
50  // Funcin que ser ejecutada por cada hilo
51  void calcularTrapecios(const Funcion& f, double x_inicio, double x_fin, double dx,
        double* resultado, std::mutex& mtx) {
52      double area_parcial = 0.0;
53
54      // Calcular el rea del trapecio en este segmento
55      double lado1 = f.evaluar(x_inicio);
56      double lado2 = f.evaluar(x_fin);
57      area_parcial = areaTrapecio(lado1, lado2, dx);
58
59      // Usar mutex para evitar race conditions al sumar el resultado
60      std::lock_guard<std::mutex> lock(mtx);
61      *resultado += area_parcial;
62  }
63
64  int main() {
65      Funcion f([](double x) { return 2*x*x + 3*x + 0.5; }, "f(x) = x"); // Funcin
            cuadrtica para mejor visualizacin
66
67      std::cout << "=== Aproximacin de Integral usando Regla del Trapecio con Hilos ==="
            << std::endl;
```

```
68
69      double x1 = 2.0; // Lmite inferior
70      double x2 = 20.0; // Lmite superior
71      f.mostrar();
72      std::cout << "Lmites de integracin: [" << x1 << ", " << x2 << "]" << std::endl;
73      std::cout << "Nmero de hilos: " << CANTIDAD_MAXIMA_DE_HILOS << std::endl <<
            std::endl;
74
75      double dx = (x2 - x1) / (double)CANTIDAD_MAXIMA_DE_HILOS;
76
77      double* resultado_integral = new double(0.0); // Memoria dinmica para el resultado
78      std::mutex mtx; // Mutex para evitar race conditions
79      std::vector<std::thread> hilos; // Vector para almacenar los hilos
80      hilos.reserve(CANTIDAD_MAXIMA_DE_HILOS); // Reservar memoria para eficiencia
81
82      auto puntos = generarPuntos(x1, x2, CANTIDAD_MAXIMA_DE_HILOS);
83
84      // Tiempo de inicio
85      auto inicio = std::chrono::high_resolution_clock::now();
86
87      // CREACIN DE HILOS - vamos a dividir el rea en trapecios
88      std::cout << "Creando y ejecutando " << CANTIDAD_MAXIMA_DE_HILOS << " hilos..." <<
            std::endl;
89      for(int i = 0; i < CANTIDAD_MAXIMA_DE_HILOS; i++){
90          double x_inicio = puntos[i];
91          double x_fin = puntos[i+1];
92
93          // Crear hilo que calcular el trapecio en este segmento
94          hilos.emplace_back(calcularTrapecios, std::ref(f), x_inicio, x_fin, dx,
                resultado_integral, std::ref(mtx));
95
96          std::cout << "Hilo " << i+1 << ": calculando trapecio en [" << x_inicio << ", "
                << x_fin << "]" << std::endl;
97      }
98
99      // EJECUCIN PARALELA - esperar a que todos los hilos terminen
100     std::cout << "\nEsperando que todos los hilos terminen su ejecucin..." << std::endl;
101     for(auto& hilo : hilos) {
102         hilo.join(); // Esperar a que cada hilo termine
103     }
104
105     auto fin = std::chrono::high_resolution_clock::now();
106     auto duracion = std::chrono::duration_cast<std::chrono::microseconds>(fin - inicio);
107
108     // Mostrar resultados
109     std::cout << "\n=== RESULTADOS ===" << std::endl;
110     std::cout << "Aproximacin de la integral: " << *resultado_integral << std::endl;
111     std::cout << "Tiempo de ejecucin: " << duracion.count() << " microsegundos" <<
            std::endl;
112
113     // Para f(x) = x, la integral de 0 a 2 es:  xdx  =  x /3 = 8/3    2.667
114     double valor_exacto = (pow(x2, 3) - pow(x1, 3)) / 3.0;
115     double error = abs(*resultado_integral - valor_exacto);
116     std::cout << "Valor exacto: " << valor_exacto << std::endl;
117     std::cout << "Error absoluto: " << error << std::endl;
118     std::cout << "Error relativo: " << (error/valor_exacto)*100 << "%" << std::endl;
```

```
119
120     // Liberar memoria dinmica
121     delete resultado_integral;
122
123     return 0;
124 }
```

- Salida de Trapecio.cpp

```
Hilo 8: calculando trapecio en [8.3, 9.2]
Hilo 9: calculando trapecio en [9.2, 10.1]
Hilo 10: calculando trapecio en [10.1, 11]
Hilo 11: calculando trapecio en [11, 11.9]
Hilo 12: calculando trapecio en [11.9, 12.8]
Hilo 13: calculando trapecio en [12.8, 13.7]
Hilo 14: calculando trapecio en [13.7, 14.6]
Hilo 15: calculando trapecio en [14.6, 15.5]
Hilo 16: calculando trapecio en [15.5, 16.4]
Hilo 17: calculando trapecio en [16.4, 17.3]
Hilo 18: calculando trapecio en [17.3, 18.2]
Hilo 19: calculando trapecio en [18.2, 19.1]
Hilo 20: calculando trapecio en [19.1, 20]

Esperando que todos los hilos terminen su ejecución...

=== RESULTADOS ===
Aproximación de la integral: 5935.86
Tiempo de ejecución: 1730 microsegundos
Valor exacto: 2664
Error absoluto: 3271
Error relativo: 122.785%
```

- Uso de Pool de threads en C++ para resolver el problema con el metodo del trapecio

Listing 4: TrapecioPool.cpp

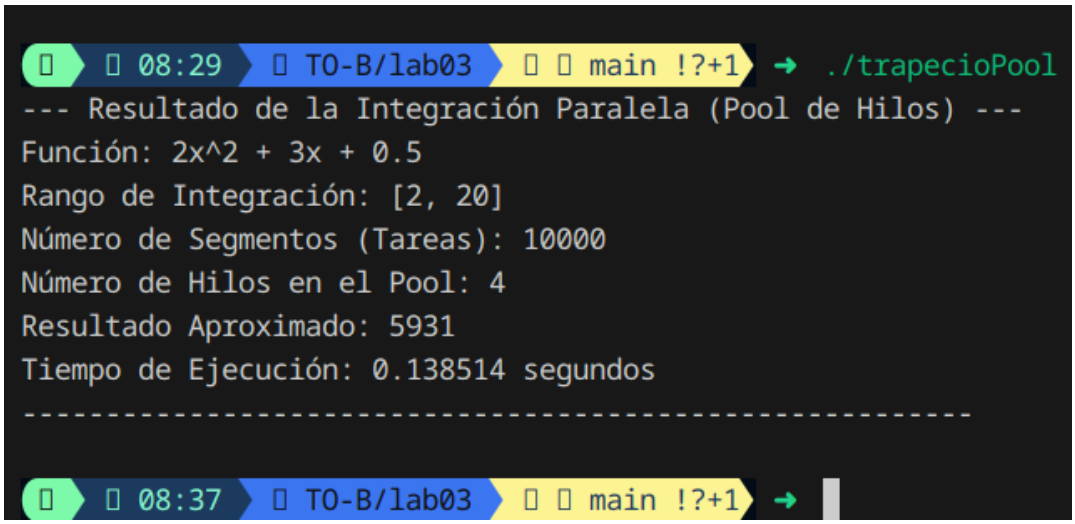
```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <queue>
#include <functional>
#include <numeric>
#include <chrono>

// Funcin  a integrar: f(x) = 2x^2 + 3x + 0.5
double function_to_integrate(double x) {
```

```
    return 2.0 * x * x + 3.0 * x + 0.5;
}

// ------------------- Thread Pool Implementation -------------------

class ThreadPool {
public:
    ThreadPool(size_t num_threads) : stop_all(false) {
        for (size_t i = 0; i < num_threads; ++i) {
            workers.emplace_back([this] {
                while (true) {
                    std::function<void()> task;
                    {
                        std::unique_lock<std::mutex> lock(queue_mutex);
                        // Espera hasta que haya una tarea o se solicite la detencin
                        condition.wait(lock, [this]{
                            return stop_all || !tasks.empty();
                        });

                        if (stop_all && tasks.empty()) {
                            return; // El hilo termina
                        }
                        task = std::move(tasks.front());
                        tasks.pop();
                    }
                    task(); // Ejecuta la tarea
                }
            });
        }
    }

    // Agrega una nueva tarea a la cola
    void enqueue(std::function<void()> task) {
        {
            std::unique_lock<std::mutex> lock(queue_mutex);
            if (stop_all) return; // No aceptar tareas si ya se est deteniendo
            tasks.emplace(std::move(task));
        }
        condition.notify_one(); // Notifica a un hilo que hay una tarea nueva
    }

    // Destructor: espera a que todos los hilos terminen
    ~ThreadPool() {
        {
            std::unique_lock<std::mutex> lock(queue_mutex);
            stop_all = true;
        }
        condition.notify_all(); // Despierta a todos los hilos
        for (std::thread &worker : workers) {
            if (worker.joinable()) {
                worker.join(); // Espera a que cada hilo termine
            }
        }
    }

private:
```

```
70      std::vector<std::thread> workers;
71      std::queue<std::function<void()>> tasks;
72      std::mutex queue_mutex;
73      std::condition_variable condition;
74      bool stop_all;
75  };
76
77  // ------------------- Integracin Paralela -------------------
78
79  // Variables globales para el clculo (necesitan ser sincronizadas)
80  double total_result = 0.0;
81  std::mutex result_mutex;
82
83  // Tarea para calcular la contribucin de un segmento (trapecio)
84  void integrate_segment(double a, double b, double h) {
85      double area = h * (function_to_integrate(a) + function_to_integrate(b)) / 2.0;
86
87      // Suma la contribucin al resultado total de forma segura
88      std::lock_guard<std::mutex> lock(result_mutex);
89      total_result += area;
90  }
91
92  int main() {
93      const double A = 0.0;     // Lmite inferior
94      const double B = 10.0;    // Lmite superior
95      const int N_TASKS = 10000; // Nmero de segmentos (tareas)
96      const int N_THREADS = 4; // Nmero de hilos en el pool
97
98      // Ancho de cada segmento
99      double h = (B - A) / N_TASKS;
100
101     // Inicializa el pool de hilos
102     ThreadPool pool(N_THREADS);
103
104     auto start_time = std::chrono::high_resolution_clock::now();
105
106     // Divide la integral en N_TASKS y las enva al pool
107     for (int i = 0; i < N_TASKS; ++i) {
108         double a_i = A + i * h;
109         double b_i = A + (i + 1) * h;
110
111         // Encola la tarea para integrar el segmento [a_i, b_i]
112         pool.enqueue([a_i, b_i, h] {
113             integrate_segment(a_i, b_i, h);
114         });
115     }
116
117     // El destructor del ThreadPool esperar automticamente a que todas
118     // las tareas encoladas se completen antes de que main termine.
119
120     // Pequeo retardo para asegurar que los hilos terminen antes de calcular el tiempo
        final
121     // En una aplicacin real, se usaran futuros/promesas o una barrera de sincronizacin,
122     // pero para este ejemplo, el destructor del ThreadPool cumple esta funcin.
123     // Simplemente aseguramos que los hilos tienen tiempo de tomar y completar la ltima
        tarea.
```

```
124     // Nota: El destructor de 'pool' se llama al final del 'main' y espera a los hilos.
125
126     // Para obtener el tiempo correcto, necesitamos esperar explcitamente a que todas
            las tareas
127     //  estn  completas antes de salir del mbito de 'pool'.
128
129     // Una forma simple de esperar es hacer que el ThreadPool rastree el nmero de
            tareas activas
130     // o simplemente mover el pool al final del bloque main para que su destructor se
            ejecute.
131
132     // Para este ejemplo, podemos confiar en que la finalizacin de 'pool' al final de
            'main'
133     // nos da el tiempo hasta que se completa la ltima tarea.
134
135     // (La temporizacin es imprecisa sin una barrera o futuros)
136
137     // Detencin  forzada  aqu  para  demostrar el tiempo despus de que se supone que todo
            termin
138     std::this_thread::sleep_for(std::chrono::milliseconds(100));
139
140     auto end_time = std::chrono::high_resolution_clock::now();
141     std::chrono::duration<double> duration = end_time - start_time;
142
143
144     // Valor terico de la integral definida de 2x^2 + 3x + 0.5 en [0, 10]:
145     // [ (2/3)x^3 + (3/2)x^2 + (1/2)x ] en [0, 10]
146     // = (2/3)(1000) + (3/2)(100) + (1/2)(10)
147     // = 666.6666... + 150 + 5 = 821.6666...
148
149     std::cout << "--- Resultado de la Integracin Paralela (Pool de Hilos) ---" <<
            std::endl;
150     std::cout << "Funcin: 2x^2 + 3x + 0.5" << std::endl;
151     std::cout << "Rango de Integracin: [" << A << ", " << B << "]" << std::endl;
152     std::cout << "Nmero de Segmentos (Tareas): " << N_TASKS << std::endl;
153     std::cout << "Nmero de Hilos en el Pool: " << N_THREADS << std::endl;
154     std::cout << "Resultado Aproximado: " << total_result << std::endl;
155     std::cout << "Tiempo de Ejecucin: " << duration.count() << " segundos" << std::endl;
156     std::cout << "------------------------------------------------------------" <<
            std::endl;
157
158     return 0;
159 }
```

- Salida de TrapecioPool.cpp



5.3. Lenguaje de Programacion - Go

- Uso de threads en Go para resolver el problema con el metodo del trapecio

Listing 5: Trapecio.go

```go
package main

import (
   "fmt"
   "math"
)

func funcion(x float64) float64 {
   return 2*x*x + 3*x + 0.5
}

// Goroutine que calcula el rea de un subintervalo con 1 trapecio
func trapecio(a, b float64, ch chan float64) {
   area := (b - a) * (funcion(a) + funcion(b)) / 2.0
   ch <- area
}

func main() {
   a, b := 2.0, 20.0
   subdivisiones := 1
   areaAnterior := -1.0
   detener := false

   for !detener {
      // Canal para recolectar resultados de cada goroutine
      ch := make(chan float64, subdivisiones)
      intervalo := (b - a) / float64(subdivisiones)

      for i := 0; i < subdivisiones; i++ {
         ai := a + float64(i)*intervalo
```

```
31          bi := ai + intervalo
32          if i == subdivisiones-1 {
33             bi = b
34          }
35          go trapecio(ai, bi, ch)
36       }
37
38       areaTotal := 0.0
39       for i := 0; i < subdivisiones; i++ {
40          areaTotal += <-ch
41       }
42
43       fmt.Printf("N = %d -> rea aproximada = %.6f\n", subdivisiones, areaTotal)
44
45       areaRedondeada := math.Round(areaTotal*10000) / 10000
46       areaAnteriorRedondeada := math.Round(areaAnterior*10000) / 10000
47
48       if areaRedondeada == areaAnteriorRedondeada {
49          detener = true
50          fmt.Println("\nSe detuvo porque el rea ya no cambia en 4 decimales - Pool de
                Threads")
51       }
52
53       areaAnterior = areaTotal
54       subdivisiones++
55    }
56 }
```

- Salida de Trapecio.go

```
N = 337 -> Área aproximada = 5931.017117
N = 338 -> Área aproximada = 5931.017016
N = 339 -> Área aproximada = 5931.016916
N = 340 -> Área aproximada = 5931.016817
N = 341 -> Área aproximada = 5931.016718
N = 342 -> Área aproximada = 5931.016620
N = 343 -> Área aproximada = 5931.016524
N = 344 -> Área aproximada = 5931.016428
N = 345 -> Área aproximada = 5931.016333
N = 346 -> Área aproximada = 5931.016238
N = 347 -> Área aproximada = 5931.016145
N = 348 -> Área aproximada = 5931.016052

Se detuvo porque el área ya no cambia en 4 decimales.
```

- Uso de Pool de threads en Go para resolver el problema con el metodo del trapecio

Listing 6: TrapecioPool.go

```go
package main

import (
   "fmt"
   "math"
)

// f(x) = 2x^2 + 3x + 0.5
func funcion(x float64) float64 {
   return 2*x*x + 3*x + 0.5
}

type tarea struct {
   a, b float64
}

func worker(id int, tareas <-chan tarea, resultados chan<- float64) {
   for t := range tareas {
      area := (t.b - t.a) * (funcion(t.a) + funcion(t.b)) / 2.0
      resultados <- area
   }
}

func main() {
   a, b := 2.0, 20.0
   subdivisiones := 1
   areaAnterior := -1.0
   detener := false

   // Nmero fijo de workers en el pool
```

```
   numWorkers := 4

   for !detener {
      tareas := make(chan tarea, subdivisiones)
      resultados := make(chan float64, subdivisiones)

      // Lanzamos los workers
      for w := 0; w < numWorkers; w++ {
         go worker(w, tareas, resultados)
      }

      // Dividimos el intervalo en "subdivisiones" partes y mandamos tareas
      intervalo := (b - a) / float64(subdivisiones)
      for i := 0; i < subdivisiones; i++ {
         ai := a + float64(i)*intervalo
         bi := ai + intervalo
         if i == subdivisiones-1 {
            bi = b
         }
         tareas <- tarea{a: ai, b: bi}
      }
      close(tareas)

      areaTotal := 0.0
      for i := 0; i < subdivisiones; i++ {
         areaTotal += <-resultados
      }

      fmt.Printf("N = %d -> rea aproximada = %.6f\n", subdivisiones, areaTotal)

      areaRedondeada := math.Round(areaTotal*10000) / 10000
      areaAnteriorRedondeada := math.Round(areaAnterior*10000) / 10000

      if areaRedondeada == areaAnteriorRedondeada {
         detener = true
         fmt.Println("\nSe detuvo porque el rea ya no cambia en 4 decimales.")
      }

      areaAnterior = areaTotal
      subdivisiones++
   }
}
```

bits de datos, 2 bit de start y 2 bit de stop? ¿Cuánta información se transmite en un día si entre carácter y carácter debe haber 2 mseg de silencio?

4. Hacer la descripción física y funcional de los siguientes puertos:

- Salida de Trapecio.go

```
64                if areaRedondeada == areaAnteriorRedondeada {
65                        detener = true
66                        fmt.Println("\nSe detuvo porque el área ya no cambia en 4 decimales - Pool de Threads")
67                }
68
69                areaAnterior = areaTotal
70                subdivisiones++
71        }
72 }
73
N = 337 -> Area aproximada = 5931.017117
N = 338 -> Área aproximada = 5931.017016
N = 339 -> Área aproximada = 5931.016916
N = 340 -> Área aproximada = 5931.016817
N = 341 -> Área aproximada = 5931.016718
N = 342 -> Área aproximada = 5931.016620
N = 343 -> Área aproximada = 5931.016524
N = 344 -> Área aproximada = 5931.016428
N = 345 -> Área aproximada = 5931.016333
N = 346 -> Área aproximada = 5931.016238
N = 347 -> Área aproximada = 5931.016145
N = 348 -> Área aproximada = 5931.016052

Se detuvo porque el área ya no cambia en 4 decimales - Pool de Threads

Program exited.
```

5.4. Estructura de laboratorio 03

- El contenido que se entrega en este laboratorio es el siguiente:

```
lab03/
|--- README.md
|--- Trapecio.java
|--- trapecio.cpp
|--- Trapecio.go
|--- TrapecioPool.java
|--- TrapecioPool.go
|--- TrapecioPool.cpp
|--- latex
   |--- img
   |   |--- logo_abet.png
   |   |--- logo_episunsa.png
   |   |--- TrapecioJava.jpg
   |   |--- TrapecioGo.jpg
   |   |--- TrapecioGoPool.jpg
   |   |--- TrapecioJavaPool.jpg
   |   |--- logo_unsa.jpg
   |--- teo_lab02_shanccom.pdf
   |--- teo_lab02_shanccom.tex
   |--- src
       |--- Trapecio.java
       |--- trapecio.cpp
       |--- Trapecio.go
       |--- TrapecioPool.java
       |--- TrapecioPool.go
       |--- TrapecioPool.cpp
```

6. Pregunta 1: ¿Cuál de los LP posee ventajas para programar paralelamente?

Respuesta

Cada lenguaje ofrece ventajas distintas en programación paralela. **Go** destaca por su modelo nativo de concurrencia mediante *goroutines* y canales, siendo más ligero y eficiente. **Java** ofrece un buen balance gracias a sus librerías maduras como *ExecutorService* y *ForkJoinPool*. **C++** brinda máximo control y rendimiento, pero exige mayor complejidad en la gestión de memoria y sincronización. Finalmente como opinion personal el lenguaje con más ventajas para la programación paralela práctica es **Go**.

7. Pregunta 2: Realizar un cuadro comparativo entre las implementaciones del patron de diseño pool of threads entre las lengujaes de programcion vistos

Característica	**C++**	**Java**	**Go**
Manejo de Hilos	*std::thread*, *std::async*	Clases en el paquete *java.util.concurrent*, como *ExecutorService* y *ThreadPoolExecutor*.	*Goroutines*, gestionadas por el *scheduler* de Go.
Sincronización	*std::mutex*, *std::condition_variable*, *std::future*. Requiere gestión manual y cuidadosa para evitar *deadlocks*.	*synchronized*, *ReentrantLock*, *Semaphore*, *Atomic* y *Concurrent Collections*. Abstracciones de alto nivel.	Canales (*channels*) para la comunicación y sincronización. Mecanismo nativo y seguro de concurrencia.
Complejidad	**Alta.** El programador debe gestionar el ciclo de vida de los hilos, la memoria y la sincronización de manera explícita, lo que ofrece máximo control pero mayor riesgo de errores.	**Media.** Las APIs de concurrencia (como *ExecutorService*) simplifican la creación y gestión del pool, pero el programador aún debe manejar la sincronización.	**Baja.** El modelo de *goroutines* y canales es intrínsecamente más simple y seguro, abstrayendo gran parte de la complejidad de la gestión de hilos.
Rendimiento	**Máximo.** Por su bajo nivel, permite optimizaciones finas del hardware y la memoria. Es ideal para tareas de alta computación donde cada nanosegundo cuenta.	**Alto.** El JIT (*Just-In-Time*) Compiler optimiza el código en tiempo de ejecución. El rendimiento es muy bueno para la mayoría de las aplicaciones.	**Excelente.** Las *goroutines* son extremadamente ligeras (pocos KB), permitiendo miles o millones de tareas concurrentes con un bajo costo de cambio de contexto.
Estilo de Código	Orientado a la gestión explícita de recursos y la programación de bajo nivel.	Orientado a objetos, con un enfoque en APIs y clases para la concurrencia.	Orientado a la concurrencia a través de la comunicación (*Don't communicate by sharing memory, share memory by communicating*).

8. Rúbricas

8.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

8.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	3	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		18	

9. Referencias

- https://www.geeksforgeeks.org/scala/scala-programming-language/
- https://onecompiler.com/scala
- https://docs.scala-lang.org/es/tour/tour-of-scala.html
- https://www.programiz.com/scala/online-compiler/
- https://www.jdoodle.com/compile-scala-online
- https://paiza.io/es/projects/new