	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Dra. Rocío Alejandra Aldeco Pérez

Asignatura: Programación orientada a objetos -1323

Grupo: 6

No de Práctica(s): 12

Integrante(s): Mendoza Hernández Carlos Emiliano

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: 2023-1

Fecha de entrega: 22 de noviembre del 2022

Observaciones:

CALIFICACIÓN: _____

Práctica 12.

Hilos.

OBJETIVOS

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

ACTIVIDADES

- Implementar hilos usando la clase *Thread*.
- Implementar hilos usando la clase *Runnable*

INTRODUCCIÓN

Imaginemos una aplicación departamental que deba realizar varias operaciones complejas. Sus funciones son descargar el catálogo de precios de los productos nuevos, realizar la contabilidad del día anterior y aplicar descuentos a productos existentes.

En un flujo normal las tareas se realizarían de forma secuencial, es decir, las tareas se ejecutarán una después de la otra. Sin embargo, si la descarga de productos nuevos tarda demasiado, los descuentos no de podrían aplicar hasta que ese proceso termine y, si se requiere un producto con descuento, éste no se podrá aplicar.

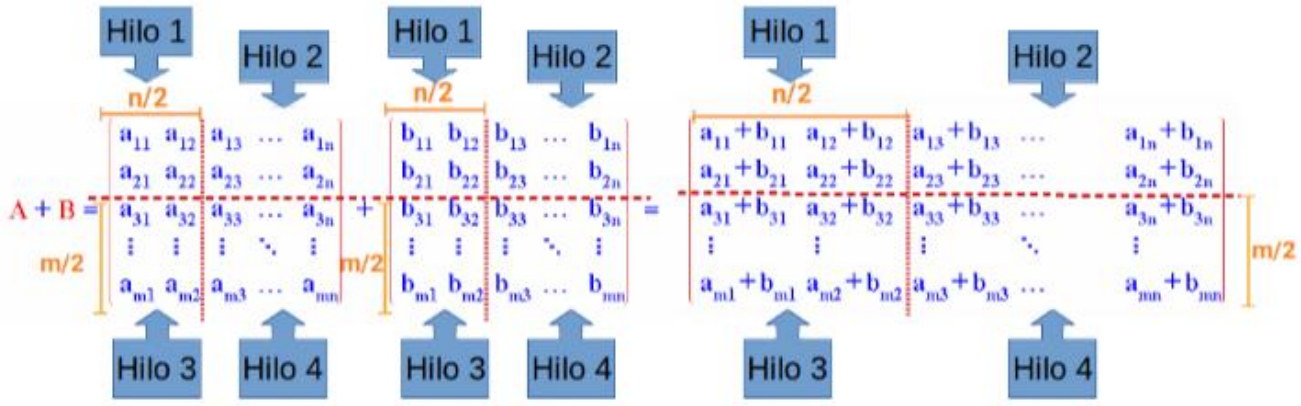
Lo ideal sería tener **varios flujos de ejecución** para poder realizar una tarea sin necesidad de esperar a las otras. Esto se puede lograr utilizando hilos, donde cada hilo representaría una tarea.

INSTRUCCIONES

Realiza las siguientes actividades después de leer y revisar en clase la *Práctica de Estudio 12: Hilos.*

Para esta práctica deberás realizar un programa que realiza la suma de dos matrices de la misma dimensión usando cuatro hilos.

1. Primero deberás de leer las matrices. Se proporcionará una fila por línea de entrada. Recuerda, las matrices son cuadradas.
2. Posteriormente deberás asignar un cuarto del trabajo a cada hilo, de tal forma que al tener 4 hilos se distribuyan el trabajo como se muestra en la imagen.
3. La suma la guardarás en una nueva matriz y la imprimirás con el mismo formato que fue leída.



DESARROLLO

Descripción de actividades realizadas

Para trabajar en esta práctica se creó el proyecto *Practica12MendozaEmiliano*. Dentro del proyecto se creó el paquete *mx.unam.fi.poo.practica12*, que es donde se creará la clase *Hilo*.

1. **Lectura de datos del programa.**

Entradas:

Para el funcionamiento de este programa se considerará el siguiente formato de números, donde m es el número de filas y n es el número de columnas.

$m \quad n$

$A_{0,0} \quad A_{0,1} \quad A_{0,2} \quad \dots \quad A_{0,n}$
$A_{1,0} \quad A_{1,1} \quad A_{1,2} \quad \dots \quad A_{1,n}$
$A_{2,0} \quad A_{2,1} \quad A_{2,2} \quad \dots \quad A_{2,n}$
$\vdots$
$A_{m,0} \quad A_{m,1} \quad A_{m,2} \quad \dots \quad A_{m,n}$

$B_{0,0} \quad B_{0,1} \quad B_{0,2} \quad \dots \quad B_{0,n}$
$B_{1,0} \quad B_{1,1} \quad B_{1,2} \quad \dots \quad B_{1,n}$
$B_{2,0} \quad B_{2,1} \quad B_{2,2} \quad \dots \quad B_{2,n}$
$\vdots$
$B_{m,0} \quad B_{m,1} \quad B_{m,2} \quad \dots \quad B_{m,n}$

Considerando lo anterior, el programa empieza leyendo las dos variables *m* y *n* en la clase *Main*.

```
int m, n;
m = stdin.nextInt();
n = stdin.nextInt();
```

Luego, conociendo las dimensiones de las matrices (dado que son del mismo tamaño), se empiezan a leer los números de las matrices en dos arreglos de enteros de dos dimensiones (uno para cada matriz, *m1* y *m2*).

```
int m1[][] = new int[m][n];
int m2[][] = new int[m][n];

for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            m1[i][j] = stdin.nextInt();
        }
}

for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            m2[i][j] = stdin.nextInt();
        }
}
```

2. **Clase *Hilo*.**

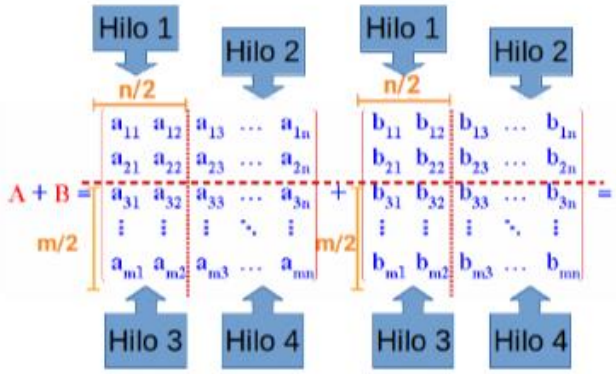
En este caso, no se pretende que la clase *Hilo* herede de alguna otra clase. Es por esta razón que elegir la implementación que hereda de la clase *Thread* es lo más ideal en vez de usar la interfaz *Runnable*.

```
public class Hilo extends Thread {
```

Atributos:

```
private int num;
private static int m;
private static int n;
private static int[][] m1;
private static int[][] m2;
private static int[][] res;
```

num: Cada hilo creado en esta clase tendrá este atributo como identificador para la parte de la matriz que le corresponde. Por ejemplo:



Si *num* = 1, ese hilo representa la parte superior izquierda de la matriz. Para *num* = *2*, ese hilo representa la parte superior derecha y así sucesivamente.

m, n: Los hilos sumarán las entradas de la matriz según los rangos que le correspondan. Se usan estos atributos (estáticos porque deben ser iguales para cada hilo creado con esta clase) para delimitar las regiones donde se estarán sumando las entradas. Se delimitan las siguientes regiones por hilo:

Hilo 1: Desde 0 hasta *m/2* para las filas. Desde 0 hasta *n/2* para las columnas.

Hilo 2: Desde 0 hasta *m/2* para las filas. Desde *n/2* hasta *n* para las columnas.

Hilo 3: Desde *m/2* hasta *m* para las filas. Desde 0 hasta *n/2* para las columnas.

Hilo 4: Desde *m/2* hasta *m* para las filas. Desde *n/2* hasta *n* para las columnas.

m1, m2: Son las matrices que se sumarán. De igual manera deben ser estáticas porque para cada hilo deben ser las mismas matrices.

res: Es una matriz de tamaño *mxn* donde se guardará la suma de *m1* y *m2*.

Métodos:

La clase tiene el siguiente **constructor**:

```
public Hilo(String nombre, int num) {
    super(nombre);
    this.num = num;
}
```

Sirve para crear un hilo y, con el parámetro *num* se indicará qué parte de la matriz se va a manejar con este hilo.

El método ***setMatrices*** funcionará como un tipo de setter para establecer las variables estáticas de la clase (tamaño de las matrices y las matrices en sí). Además, inicializa la matriz *res* como un arreglo de dos dimensiones de enteros.

```
public static void setMatrices (int m, int n, int[][] m1, int[][] m2) {
    Hilo.m = m;
    Hilo.n = n;
    Hilo.m1 = m1;
    Hilo.m2 = m2;
    res = new int [m][n];
}
```

El método más importante de esta clase es el método ***run***. Lo que se contenga en este método se ejecutará cuando se invoque al método *start*. El método ***run*** de esta clase se auxilia del atributo *num* para entrar en una estructura *switch-case*. Para cada *case* se suman las entradas de la matriz dentro de la región que le corresponde al número del hilo (atributo *num*). La suma de las entradas se almacena en su correspondiente posición de la matriz *res*.

```java
public void run() {
    switch (num) {
        case 1:
            for (int i = 0; i < m / 2; i++) {
                for (int j = 0; j < n / 2; j++) {
                    res[i][j] = m1[i][j] + m2[i][j];
                }
            }
            break;

        case 2:
            for (int i = 0; i < m / 2; i++) {
                for (int j = n / 2; j < n; j++) {
                    res[i][j] = m1[i][j] + m2[i][j];
                }
            }
            break;

        case 3:
            for (int i = m / 2; i < m; i++) {
                for (int j = 0; j < n / 2; j++) {
                    res[i][j] = m1[i][j] + m2[i][j];
                }
            }
            break;

        case 4:
            for (int i = m / 2; i < m; i++) {
                for (int j = n / 2; j < n; j++) {
                    res[i][j] = m1[i][j] + m2[i][j];
                }
            }
            break;

        default:
            System.out.println("Lo siento. El numero que ingresaste no es válido.");
            break;
    }
}
```

Para imprimir el resultado, su creó el método ***printSum***.

```
public static void printSum() {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            System.out.print(res[i][j] + " ");
        }
        System.out.println();
    }
}
```

3. Ejecución de los hilos en la clase *Main.*

Después de haber leído las entradas del programa, se hace lo siguiente:

1) Se establecen las variables estáticas de la clase.
2) Se crean los 4 hilos, indicando en cada uno el número que le corresponde.
3) Se imprime la matriz resultada.

```
Hilo.setMatrices(m, n, m1, m2);
new Hilo("Hilo 1", 1).start();
new Hilo("Hilo 2", 2).start();
new Hilo("Hilo 3", 3).start();
new Hilo("Hilo 4", 4).start();
Hilo.printSum();
```

4. Probando con diferentes entradas

input
Entrada:

```
2 3

4 3 0
-1 6 1

1 0 1
4 3 -2
```

input2
Entrada:

```
3 3

2 0 1
3 0 0
5 1 1

1 0 1
1 2 1
1 1 0
```

input3
Entrada:

```
3 3

1 4 7
2 5 8
3 6 9

1 -1 2
2 -1 2
3 -3 0
```

input4
Entrada:

```
6 8

1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1

8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8
8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8
```

Salidas:

CONCLUSIONES

En esta práctica se implementó el manejo de hilos utilizando la clase *Thread*. Es importante destacar que existe otra opción en *Java* para el manejo de hilos que es usando la interfaz *Runnable*. Para este caso se utilizó la clase *Thread*, sin embargo, en una clase que ya establece una relación de herencia, la interfaz *Runnable* permite el manejo de hilos. Se dividió el flujo del programa, asignando a 4 hilos una región de las matrices que se sumaron. Los resultados obtenidos fueron los esperados, por lo tanto, considero que se cumplieron los objetivos de la práctica.

REFERENCIAS

- *Deitel Paul, Deitel Harvey*
  Cómo programar en Java.
  Séptima Edición.
  México
  Pearson Educación, 2008

- *Martín, Antonio*
  Programador Certificado Java 2.
  Segunda Edición.
  México
  Alfaomega Grupo Editor, 2008

- *Dean John, Dean Raymond*
  Introducción a la programación con Java.
  Primera Edición.
  México
  Mc Graw Hill, 2009

- *Sierra Katy, Bates Bert*
  SCJP Sun Certified Programmer for Java 6 Study Guide.
  Mc Graw Hill

Yo, Carlos Emiliano Mendoza Hernández, hago mención que esta práctica fue de mi autoría.