**1. Identificación.**

|  |
| --- |
| **NOMBRE DE LA PRÁCTICA: PROGRAMACIÓN CONCURRENTE No. DE PRÁCTICA:** 7 **No. DE SESIONES:** 3 **No. DE INTEGRANTES MÁXIMO POR EQUIPO:** 1 |

**2. Introducción.**

Un Lenguaje de Programación será concurrente si posee las estructuras necesarias para definir y manejar diferentes tareas (hilos de ejecución) dentro de un programa.

La concurrencia es, en esencia, el poder realizar múltiples cosas en el mismo tiempo, pero, no específicamente en paralelo.

Una de las formas más sencillas de comprender la concurrencia es imaginar a una persona la cual trabaja en múltiples tareas al mismo tiempo, y que rápidamente cambia de una tarea a otra. Por ejemplo, imaginemos a una persona la cual se encuentra programando, realizando cálculos en Excel y contestando correos electrónicos, todo esto al mismo tiempo. Dedica un par de segundos a cada tarea, y rápidamente, con un ágil cmd + shift cambia de tarea.

Concluimos que la persona trabaja de forma concurrente. Las tareas que realiza no necesariamente deben seguir un orden, quizás, después de contestar un correo regresa con los cálculos en Excel, le dedica un par de segundo, regresa a responder otro correo y finaliza con la codificación del programa, u, otro escenario pudiera ser que después finalizar ciertos cálculos, la persona continúa codificando un par de segundos para después responder un par de correos y regresar con los cálculos.

En el siguiente ejemplo se realizan 3 tareas al mismo tiempo:

|  |
| --- |
| import time  import threading  def codificar():  time.sleep(2)  print(f'Codificando')  def responder\_correos():  time.sleep(2) |

|  |
| --- |
| print(f'Respondiendo correos')  def realizar\_calculos():  time.sleep(2)  print(f'Realizar los calculos')  threading.Thread(target=codificar).start()  threading.Thread(target=responder\_correos).start()  threading.Thread(target=realizar\_calculos).start() |

En este caso el programa realiza tres tareas al mismo tiempo. A cada tarea le tomó un máximo de dos segundos ser completada, como se ejecutan de forma concurrente (al mismo tiempo) al programa le toma dos segundos finalizar su ejecución. Por otro lado, si se ejecutara el mismo código, pero ahora de forma secuencial, al programa le tomaría seis segundos finalizar. El tiempo es clave al momento de procesar instrucciones.

|  |
| --- |
| #secuencial  codificar()  responder\_correos()  realizar\_calculos() |

La principal diferencia entre concurrencia y paralelismo recae en la forma en que se realizan las tareas. Cuando se ejecutan tareas de forma concurrente a estas se les asigna un x periodo de tiempo antes de cambiar de tarea, será en ese periodo en el cual se inicie, continúe, o se complete la tarea, por otro lado, si ejecutamos tareas en paralelo, las tareas se realizarán de forma simultánea, comenzarán y finalizarán sin interrupciones.

Si lo que se quiere es implementar concurrencia en los programas una muy buena idea será utilizar Threads, por otro lado, si lo que se quiere es implementar paralelismos es mejor utilizar procesos.

**3. Objetivo General.**

|  |
| --- |
| Comprender la programación concurrente, a través de algunos ejemplos prácticos con el lenguaje de programación de Java . |

**4. Objetivos Específicos.**

|  |
| --- |
| ∙ Codificar ejemplos de programación con hilos en el lenguaje de programación de Java. |

|  |
| --- |
| ∙ Entender el concepto de concurrencia por medio de ejemplos prácticos en el lenguaje de programación de Java |

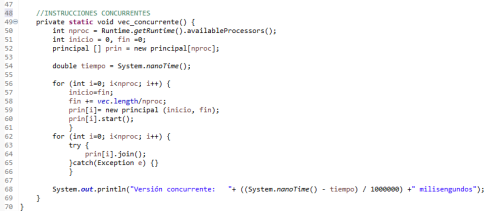
**5. Materiales/utensilios y equipos.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a) MATERIALES/UTENSILIOS.** | | | | |
| CANTIDAD | DESCRIPCIÓN | | ESPECIFICACIONES | OBS. |
|  | Equipo de cómputo con acceso a internet | | Cualquier versión |  |
|  | Eclipse instalado o Netbeans instalado | |  |  |
| **b) EQUIPOS/INSTRUMENTOS.** | | | | |
| CANTIDAD | DESCRIPCIÓN | ESPECIFICACIONES | | OBS. |
| 20 | Equipo de cómputo por alumno |  | |  |
|  |  |  | |  |

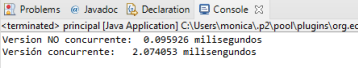
**6. Desarrollo de la Actividad Práctica.**

|  |
| --- |
| La Máquina Virtual Java (JVM) es un sistema multihilo. Es decir, es capaz de ejecutar varios hilos de ejecución simultáneamente. La JVM gestiona todos los detalles, asignación de tiempos de ejecución, prioridades, etc., de forma similar a como gestiona un Sistema Operativo múltiples procesos.  La diferencia básica entre un proceso de Sistema Operativo y un Thread Java es que los hilos corren dentro de la JVM, que es un proceso del Sistema Operativo y por tanto comparten todos los recursos, incluida la memoria y las variables y objetos allí definidos. A este tipo de procesos donde se comparte los recursos se les llama procesos ligeros (lightweight process).  Java da soporte al concepto de Thread desde el propio lenguaje, con algunas clases e interfaces definidas en el paquete java.lang y con métodos específicos para la manipulación de Threads en la clase Object.  Desde el punto de vista de las aplicaciones los hilos son útiles porque permiten que el flujo del programa sea divido en dos o más partes, cada una ocupándose de alguna tarea de forma independiente.  Por ejemplo un hilo puede encargarse de la comunicación con el usuario, mientras que otros actúan en segundo plano, realizando la transmisión de un fichero, accediendo a recursos del sistema (cargar sonidos, leer ficheros ...), etc. De hecho, todos los programas con interface gráfico (AWT o Swing) son multihilo porque los eventos y las rutinas de dibujado de las ventanas corren en un hilo distinto al principal.  En el siguiente código al ejecutarlo podemos observar el tiempo que tarda el microprocesador en procesar 3 simples operaciones utilizando la programación secuencial y en contraparte la programación concurrente. |

|  |
| --- |
|  |

****

El resultado de la ejecución del código puede variar de acuerdo a las características de cada equipo de cómputo.



Resultados obtenidos:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En la programación concurrente, se utilizan con gran frecuencia el uso de hilos (thread).

Una thread es un único flujo de control dentro de un programa. Algunas veces es llamado contexto de ejecución porque cada thread debe tener sus propios recursos, como el program counter y el stack de ejecución, como el contexto de ejecución. Sin embargo, toda thread en un programa aún comparte muchos recursos, tales como espacio de memoria y archivos abiertos. Los Threads también son llamadas procesos livianos (lightweight process).

Hay dos formas de hacer una tarea correr concurrentemente con otra: crear una nueva clase como subclase de la clase Thread o declarar una clase e implementar la interfaz Runnable.

**Uso de Subclase**

Cuando se crea una subclase de Thread, la subclase debería definir su propio método run() para sobre montar el método run() de la clase Thread. La tarea concurrente es desarrollada en este método run().

**Ejecución del método run()**

|  |
| --- |
| Una instancia de la subclase es creada con new, luego se llama al método start() de la thread para hacer que la máquina virtual Java ejecute el método run(). Para iniciar la concurrencia se invoca a start(), así se invoca a run() en forma indirecta. Si se invoca a run() directamente, se comportará como el llamado a cualquier método llamado dentro de un mismo hilo (sin crear uno independiente).  **Implementación de la Interfaz Runnable**  La interfaz Runnable requiere que sólo un método sea implementado, el método run(). Primero se crea una instancia de esta clase con new, luego se crea una instancia de Thread con otra sentencia new y se usa el objeto recién creado en el constructor. Finalmente, se hace el llamado al método start() de la instancia de Thread para iniciar la tarea definida en el método run().  El ciclo de vida de un Thread. Cada hilo, después de su creación y antes de su destrucción, estará en uno de cuatro estados:  ∙ recién creada,  ∙ "corrible",  ∙ bloqueada, o  ∙ muerta.  Ejemplo de uso de hilos  Resultado de la ejecución de código. |

Página 6

|  |
| --- |
| Codifica este ejemplo y ejecútalo varias veces y observa lo que sucede. Al ejecutar el código varias veces se puede ver que no siempre se ejecuta igual. Realiza la captura de pantalla de esas ejecuciones diferentes y documenta el proceso.    Notas sobre el programa:  ∙ La clase **Thread** está en el paquete **java.lang**. Por tanto, no es necesario el **import**. ∙ El constructor **public Thread(String str)** recibe un parámetro que es la identificación del **Thread**.  ∙ El método **run()** contiene el bloque de ejecución del **Thread**. Dentro de él, el método **getName()** devuelve el nombre del **Thread** (el que se ha pasado como argumento al constructor).  ∙ El método **main** crea dos objetos de clase **ThreadEjemplo** y los inicia con la llamada al método **start()** (el cual inicia el nuevo hilo y llama al método run()).  ∙ Obsérvese en la salida el primer mensaje de finalización del **thread main**. La ejecución de los hilos es asíncrona. Realizada la llamada al método **start()**, éste le devuelve control y continua su ejecución, independiente de los otros hilos.  ∙ En la salida los mensajes de un hilo y otro se van mezclando. La máquina virtual asigna tiempos a cada hilo. |

|  |
| --- |
| En este ejemplo de código se está utilizando la interface **Runnable** la cual proporciona un método alternativo a la utilización de la clase **Thread**, para los casos en los que no es posible hacer que la clase definida extienda la clase **Thread**. Esto ocurre cuando dicha clase, que se desea ejecutar en un hilo independiente deba extender alguna otra clase. Dado que no existe  herencia múltiple, la citada clase no puede extender a la vez la clase **Thread** y otra más. En este caso, la clase debe implantar la interface **Runnable**, variando ligeramente la forma en que se crean e inician los nuevos hilos.  Codifica este ejemplo y ejecútalo varias veces y observa lo que sucede. Al ejecutar el código varias veces se puede ver que no siempre se ejecuta igual. Realiza la captura de pantalla de esas ejecuciones diferentes y documenta el proceso. |

**7. Cuestionario.**

|  |
| --- |
| 1.- ¿Cuáles son las ventajas de utilizar la programación concurrente?   * **Hacer un mejor uso de los procesadores de computadora**   Ahora las CPU son de múltiples núcleos y cada vez más núcleos. Si un programa es de un solo subproceso, solo puede usar un núcleo de todos modos, y ninguna cantidad de núcleos de CPU puede mejorar el rendimiento de la máquina.  Si un programa es multiproceso, puede utilizar varios núcleos de la CPU para realizar operaciones. Cuantos más núcleos de la CPU, más rápido se puede ejecutar.   * **Velocidad de respuesta más rápida**   El subproceso múltiple permite ejecutar una serie de operaciones simultáneamente, lo que puede mejorar la velocidad de respuesta del programa. Por ejemplo, cuando un usuario realiza un pedido, incluye insertar datos del pedido, generar una instantánea del pedido, enviar un correo electrónico para notificar al vendedor y registrar la cantidad de bienes vendidos.  Una vez finalizadas estas operaciones comerciales, el usuario debe esperar 1 segundo. El usuario puede esperar 0,5 s para operaciones como generar instantáneas de pedidos y enviar correos electrónicos a otros subprocesos para su procesamiento.   * **Más fácil de desarrollar**   La programación concurrente se consideró al comienzo del diseño de Java y proporcionó un modelo de programación consistente.Los programadores que usan Java pueden enfocarse más en su negocio sin tener que considerar cómo usar el multiproceso.  2.- ¿Cuáles son las desventajas de utilizar la programación concurrente?   * La limitación de recursos significa que la velocidad de ejecución del programa está limitada por los recursos de hardware o software de la computadora durante la programación simultánea. * Se produce un interbloqueo, lo que hace que el sistema no esté disponible * El proceso de cambio de contexto entre hilos es así. A cada subproceso de la CPU se le asigna un intervalo de tiempo, y un intervalo de tiempo representa un período corto de tiempo. Cuando la CPU ejecuta un subproceso, cuando se agota el intervalo de tiempo, cambiará a otro subproceso para su ejecución y, por lo tanto, cambiará entre subprocesos.   3.- ¿Son iguales la programación paralela y la concurrente? NO, Explica ¿por qué? Se puede tener dos hilos (o procesos) ejecutándose simultáneamente en el mismo núcleo a través del cambio de contexto. Cuando los dos hilos (o procesos) se ejecutan en dos núcleos (o procesadores) diferentes, tiene paralelismo. Entonces, en el primer caso (concurrencia) el paralelismo es solo "virtual", mientras que en el segundo tienes un verdadero paralelismo. Por lo tanto, cada programa paralelo es concurrente, pero lo contrario no es necesariamente cierto.  4.- ¿Cuáles son las diferencias entre la programación secuencial y la programación concurrente?  La programación secuencial (también llamada estructura secuencial), es un programa que hace que una instrucción siga a otra instrucción después de que esta haya finalizado mientras que la programación concurrente consiste en utilizar técnicas de programación que hagan que múltiples instrucciones de un programa funcionen al mismo tiempo. |

**8. Referencias Bibliográficas.**

|  |
| --- |
| Moya Ricardo (2014). Multitarea e hilos en Java, disponible en:  https://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/ Vallejo D., González C. y Albusac J. (2016) Programación concurrente en tiempo real, 3ra Edición, ISBN:978-1518609261. Disponible en: |

|  |
| --- |
| http://190.57.147.202:90/xmlui/bitstream/handle/123456789/445/Programacion Concurrente-y-Tiempo-Real.pdf?sequence=1&isAllowed=y |

**9. Formato y especificación del reporte de práctica.**

|  |
| --- |
| **a) Introducción**  **a) Objetivo**  **b) Desarrollo de la actividad práctica**  **c) Resultados**  **d) Discusión**  **e) Cuestionario**  **f) Referencias Bibliográficas** |

[[1]](#footnote-1)

1. [↑](#footnote-ref-1)