

**Tema**

TALLER HILOS INTERRUPCIONES CON MVC DOTNET

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

31/12/2022

**TALLER HILOS INTERRUPCIONES CON MVC DOTNET**

[1 INTRODUCCIÓN 4](#_Toc124907232)

[2 OBJETIVOS 4](#_Toc124907233)

[2.1 OBJETIVO GENERAL 4](#_Toc124907234)

[2.2 OBJETIVOS ESPECÍFICOS 4](#_Toc124907235)

[3 MARCO TEÓRICO 4](#_Toc124907236)

[3.1 PROCESOS Y MODELOS DE CONCURRENCIA 4](#_Toc124907237)

[3.1.1 ESTADOS Y OPERACIONES DE UN PROCESO 5](#_Toc124907238)

[3.1.2 FORMAS DE CREAR Y MANEJAR PROCESOS 6](#_Toc124907239)

[3.2 HILOS 7](#_Toc124907240)

[3.2.1 CARACTERISCAS DE LOS HILOS 8](#_Toc124907241)

[3.2.2 TIPOS DE HILOS 8](#_Toc124907242)

[3.2.3 GESTIÓN DE HILOS 10](#_Toc124907243)

[3.2.4 INTERRUPCIONES 11](#_Toc124907244)

[3.2.5 HILOS CON C# 11](#_Toc124907245)

[4 PARTE PRÁCTICA 14](#_Toc124907246)

[4.1 CREACIÓN DEL PROYECTO 14](#_Toc124907247)

[4.2 ARQUITECTURA MVC PARA EL PROYECTO 16](#_Toc124907248)

[4.3 PRÁCTICA VIDEO 171 17](#_Toc124907249)

[4.3.1 CODIFICACIÓN CONTROLADOR 17](#_Toc124907250)

[4.4 PRÁCTICA VIDEO 172 23](#_Toc124907251)

[4.4.1 CODIFICACIÓN CONTROLADOR 23](#_Toc124907252)

[4.5 ESTRUCTURA DE LA APLICACIÓN 28](#_Toc124907253)

[4.6 EJECUCIÓN DEL PROYECTO 28](#_Toc124907254)

[4.6.1 EJECUCIÓN VIDEO 171 28](#_Toc124907255)

[4.6.2 EJECUCIÓN VIDEO 172 29](#_Toc124907256)

[5 CONCLUSIONES 29](#_Toc124907257)

[6 RECOMENDACIONES 29](#_Toc124907258)

[7 REFERENCIAS 30](#_Toc124907259)

**INDICE DE IMÁGENES**

[Figura 1. Procesos e Hilos 5](#_Toc124907260)

[Figura 2. Estados de un proceso 6](#_Toc124907261)

[Figura 3. Manejo de hilos 7](#_Toc124907262)

[Figura 4. Hilos y Procesos 7](#_Toc124907263)

[Figura 5. Modelo genérico de un hilo 8](#_Toc124907264)

[Figura 6. Hilos del procesador 9](#_Toc124907265)

[Figura 7. Hilos de Interfaz de Usuario 9](#_Toc124907266)

[Figura 8. Hilos de trabajo 10](#_Toc124907267)

[Figura 9. Gestión de Hilos 10](#_Toc124907268)

[Figura 10. Interrupción de hilos 11](#_Toc124907269)

[Figura 11. Logo identificativo de C# con Hilos 12](#_Toc124907270)

[Figura 12. Ventana de proyectos Visual Studio 14](#_Toc124907271)

[Figura 13. Plantillas de proyectos Visual Studio 15](#_Toc124907272)

[Figura 14. Ingreso del nombre y carpeta para el programa 15](#_Toc124907273)

[Figura 15. Selección de la version del Framework .NET 16](#_Toc124907274)

[Figura 16. Archivos generados para el programa 16](#_Toc124907275)

[Figura 17. Creación de carpetas para MVC 17](#_Toc124907276)

[Figura 18. Arquitectura MVC para el programa 17](#_Toc124907277)

[Figura 19. Creación de Clase en el controlador 18](#_Toc124907278)

[Figura 20. Asignación del nombre HiloControlador para la Clase 18](#_Toc124907279)

[Figura 21. Codificación de la Clase HiloControlador 19](#_Toc124907280)

[Figura 22. Creación de la Clase GestionHilo 21](#_Toc124907281)

[Figura 23. Codificación de la clase GestionHilo 21](#_Toc124907282)

[Figura 24. Codificación de la Clase Program 23](#_Toc124907283)

[Figura 25. Creación de la clase Hilo2Controlador 23](#_Toc124907284)

[Figura 26. Codificación de la Clase Hilo2Controlador 24](#_Toc124907285)

[Figura 27. Codificación de la Clase GestionHilo 26](#_Toc124907286)

[Figura 28. Estructura final del programa con MVC 28](#_Toc124907287)

[Figura 29. Ejecución del programa 28](#_Toc124907288)

[Figura 30. Salida de la ejecución del video 171 29](#_Toc124907289)

[Figura 31. Salida de la ejecución del video 172 29](#_Toc124907290)

**ÍNDICE DE TABLAS**

[Tabla 1. Estados de los hilos 10](#_Toc124907291)

[Tabla 3. Propiedades de la Clase Thread 12](#_Toc124907292)

[Tabla 4. Métodos de la Clase Thread 12](#_Toc124907293)

[Tabla 5. Codificación del archivo HiloControlador.cs práctica video 171 20](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA2\DOCUMENTACION\TALLER_HILOS_PRACTICA2.docx#_Toc124907294)

[Tabla 6. Codificación del archivo GestionHilo.cs práctica video 171 22](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA2\DOCUMENTACION\TALLER_HILOS_PRACTICA2.docx#_Toc124907295)

[Tabla 7. Codificación del archivo Program.cs práctica video 171 23](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA2\DOCUMENTACION\TALLER_HILOS_PRACTICA2.docx#_Toc124907296)

[Tabla 8. Codificación del archivo Hilo2Controlador.cs práctica video 172 25](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA2\DOCUMENTACION\TALLER_HILOS_PRACTICA2.docx#_Toc124907297)

[Tabla 9. Codificación del archivo GestionHilo.cs práctica video 172 27](file:///C:\Users\santi\Downloads\HILOS_DOTNET_GRUPO05\TALLER_HILOS_PRACTICA2\DOCUMENTACION\TALLER_HILOS_PRACTICA2.docx#_Toc124907298)

# INTRODUCCIÓN

Un hilo, también conocido como Thread, proceso ligero o subproceso, es la unidad de ejecución de un proceso. Está asociado con una secuencia de instrucciones, un conjunto de registros y una pila (Elorduy, 2020). Los programas que no utilizan hilos solo tienen una secuencia de ejecución y realizan una tarea a la vez. Estos programas son adecuados para tareas simples y rápidas, pero pueden ser ineficientes para tareas más complejas o que requieren la ejecución de varias tareas simultáneamente.

Los programas que utilizan hilos tienen varias secuencias de ejecución que pueden realizar tareas de manera concurrente, lo que puede mejorar la eficiencia y la velocidad de la aplicación. En C#, se pueden utilizar la clase Thread para implementar hilos, que proporciona varias funciones para crear, controlar y sincronizar hilos. Sin embargo, es importante tener en cuenta que la ejecución concurrente de varios hilos puede ser complicada y requerir una buena gestión para evitar conflictos y errores, por lo que es importante entender cómo funcionan los hilos y cómo se pueden utilizar de manera efectiva en una aplicación.

La interrupción de hilos es una técnica utilizada para detener o finalizar un hilo en ejecución. Los hilos son una característica importante en la programación concurrente, ya que permiten ejecutar varias tareas al mismo tiempo, mejorando la eficiencia y velocidad de una aplicación. Sin embargo, en algunas situaciones puede ser necesario interrumpir un hilo para liberar recursos o finalizar una tarea específica. La interrupción de hilos se realiza mediante una señal o una llamada a un método específico, y puede ser manejada por el hilo mediante un controlador de interrupciones.

# OBJETIVOS

## OBJETIVO GENERAL

Analizar los conceptos y como usar interrupciones en hilos de sistemas de software que ejecuten más de una tarea por medio de un ejemplo práctico y sencillo, utilizando el lenguaje de programación C#, la plataforma de desarrollo .NET y la arquitectura MVC (Modelo-Vista-Controlador), para implementarlos en sistemas que necesiten ejecutar varias tareas al mismo tiempo.

## OBJETIVOS ESPECÍFICOS

* Analizar los conceptos y como usar interrupciones en hilos de sistemas de software complejos que ejecuten más de una tarea.
* Identificar los pasos para implementar un ejemplo del uso de hilos con el lenguaje de programación C#, la plataforma .NET y la arquitectura MVC (Modelo-Vista-Controlador).
* Documentar todo el proceso de desarrollo del ejemplo para tener una guía cuando se necesite implementar hilos en sistemas que ejecuten varias tareas al mismo tiempo.

# MARCO TEÓRICO

## PROCESOS Y MODELOS DE CONCURRENCIA

Un proceso es una entidad compuesta por un código ejecutable que se ejecuta en un orden secuencial, un conjunto de datos y una pila que se utiliza para pasar parámetros, restaurar llamadas recursivas o interrupciones, entre otros (Rossainz, 2015).

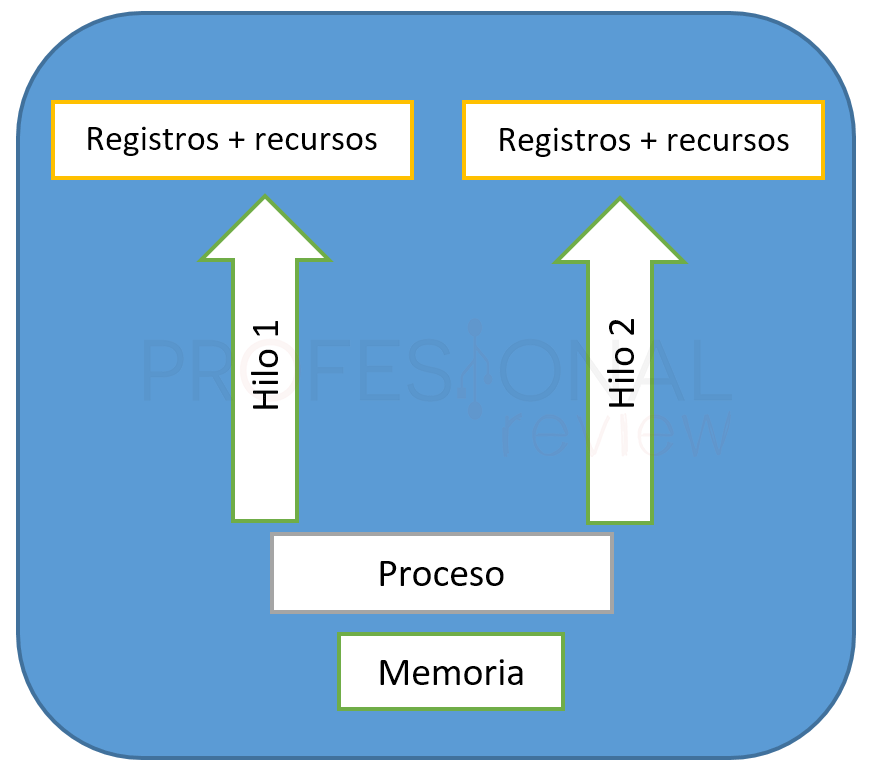


Figura . Procesos e Hilos

Aunque el concepto de proceso es similar en todos los lenguajes de programación concurrente, existen variaciones en cuanto al modelo de concurrencia utilizado. Estas variaciones se dan en aspectos como:

* La estructura:
  + Estática: cuando el número de procesos en el programa concurrente es fijo y se conoce en tiempo de compilación.
  + Dinámica: cuando los procesos pueden ser creados en cualquier momento y el número de procesos sólo se conoce en tiempo de ejecución.
* El Nivel de Paralelismo:
  + Anidado: cuando un proceso puede ser definido dentro de otro proceso.
  + Plano: cuando los procesos sólo pueden ser definidos en el nivel más externo del programa.
* Relaciones entre Procesos: Se pueden crear jerarquías de procesos e interrelaciones entre ellos con el uso de los niveles de paralelismo anidados.
  + Relación Padre/Hijo: Un proceso (el padre) es el responsable de la creación de otro, el hijo. El padre tiene que esperar mientras el hijo está siendo creado e inicializado.
  + Relación Guardian/Dependiente: La relación entre un proceso guardián y los procesos dependientes es tal que el proceso guardián no puede finalizar hasta que todos los procesos dependientes hayan terminado.
* Granularidad: Se puede dividir en dos tipos: paralelismo de grano fino y paralelismo de grano grueso. Un programa concurrente de grano grueso tiene una cantidad relativamente pequeña de procesos, cada uno con una tarea importante que realizar. Por otro lado, un programa de grano fino tiene una gran cantidad de procesos, algunos de los cuales pueden incluir solo una acción.

### **ESTADOS Y OPERACIONES DE UN PROCESO**

Un proceso puede estar en uno de varios estados diferentes durante su tiempo de existencia. El cambio de un estado a otro ocurre debido a ciertas acciones (Rossainz, 2015). Algunos de estos estados posibles incluyen: Nuevo, cuando un proceso es creado por un usuario al ejecutar un programa concurrente.

* En ejecución: cuando las instrucciones de código del proceso son ejecutadas por el procesador.
* Bloqueado: cuando el proceso está esperando a que ocurra algún evento, como una operación de entrada/salida.
* Listo: cuando el proceso está esperando a ser ejecutado por el procesador cuando el planificador del sistema operativo le dé otorgue el turno.
* Finalizado: cuando el proceso ha terminado y todos los recursos utilizados para su ejecución son liberados.

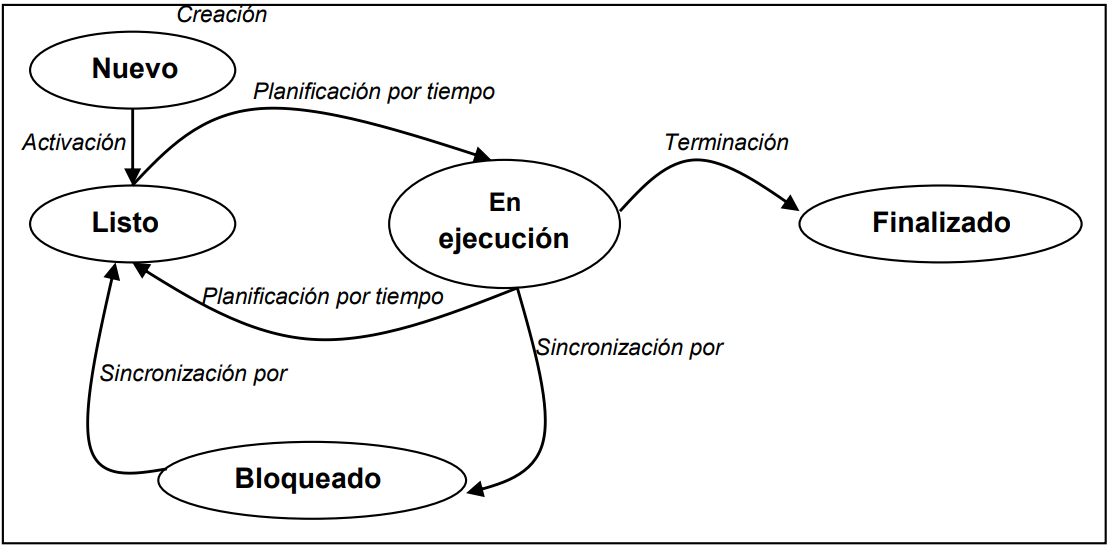


Figura . Estados de un proceso

### **FORMAS DE CREAR Y MANEJAR PROCESOS**

Existen varias formas de hacer programación concurrente, entre ellas las llamadas al sistema operativo, los lenguajes de programación que soportan programación concurrente con independencia del sistema operativo, y las bibliotecas para creación y operaciones con procesos independientes del sistema operativo.

* Las llamadas al sistema operativo son una forma de conseguir mayor eficiencia en la ejecución de los programas, pero también son consideradas como más complejas, menos obvias y con un alto costo para ser adaptadas a otro sistema operativo.
* Los lenguajes de programación que soportan programación concurrente con independencia del sistema operativo, como Ada, Occam, Pascal concurrente, C concurrente, Java, etc. son considerados como más fáciles de expresar, probar y mantener, mejoran el poder de abstracción, la modelización del mundo físico es más clara y natural, y los programas son menos dependientes de la máquina.
* Las bibliotecas para creación y operaciones con procesos independientes del sistema operativo, como Pthreads en C, son una forma de soportar la concurrencia mediante la definición de tipos y operaciones que pueden ser invocados por una aplicación. La interfaz entre el programador y dichos módulos es independiente de la máquina y del sistema operativo.

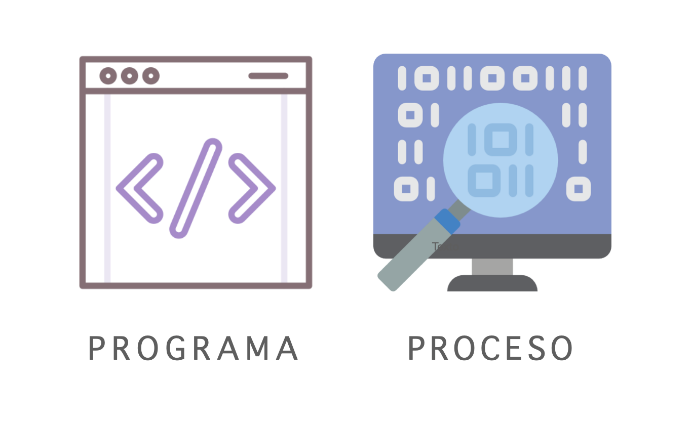


Figura . Manejo de hilos

## HILOS

Un hilo es un solo flujo de ejecución de instrucciones dentro de un proceso, debido a que no pueden ejecutarse por sí solos, requieren la supervisión de un proceso (Nakayama, 2009). Los hilos se utilizan a menudo para realizar tareas en segundo plano o para realizar tareas que deben ejecutarse de manera simultánea. Por ejemplo, si una aplicación tiene una interfaz gráfica de usuario (GUI), se puede utilizar un hilo para actualizar la interfaz mientras se realizan otras tareas en segundo plano, como descargar datos de Internet o realizar cálculos.

Timeline

Description automatically generated with medium confidence

Figura . Hilos y Procesos

En algunos lenguajes de programación, los hilos se implementan como una parte integrada del lenguaje, mientras que en otros lenguajes se pueden utilizar bibliotecas o frameworks externos para trabajar con hilos. Los hilos también se pueden utilizar para dividir el trabajo en un sistema con múltiples núcleos o procesadores, lo que permite que el trabajo se ejecute de manera más rápida y eficiente.

### **CARACTERISCAS DE LOS HILOS**

Chart

Description automatically generated

Figura . Modelo genérico de un hilo

* Los threads son más pequeños comparados con los procesos.
* La creación de un thread es relativamente menos costosa.
* Los threads comparten los recursos mientras que los procesos requieren su propio conjunto de recursos.
* Los threads ocupan menos memoria (es decir, son más económicos respecto del gasto de recursos computacionales en un sistema).
* Los threads proporcionan a los programadores la posibilidad de escribir aplicaciones concurrentes que se pueden ejecutar tanto en sistemas monoprocesador, como en sistemas multiprocesador de forma transparente.
* Los threads pueden incrementar el rendimiento en entornos monoprocesador.

### **TIPOS DE HILOS**

Los hilos son una característica importante en la programación concurrente, ya que permiten ejecutar varias tareas al mismo tiempo, mejorando la eficiencia y velocidad de una aplicación (Davizhe, 2014). Existen varios tipos de hilos, cada uno con características y usos específicos.

**HILOS DEL PROCESADOR**

Uno de los tipos de hilos es el hilo del procesador, también conocido como hilo de sistema. Este tipo de hilo está asociado directamente con el procesador y se encarga de realizar tareas críticas, como la gestión de interrupciones y la ejecución de tareas de sistema. Los hilos del procesador son administrados por el sistema operativo y tienen prioridad sobre otros hilos.

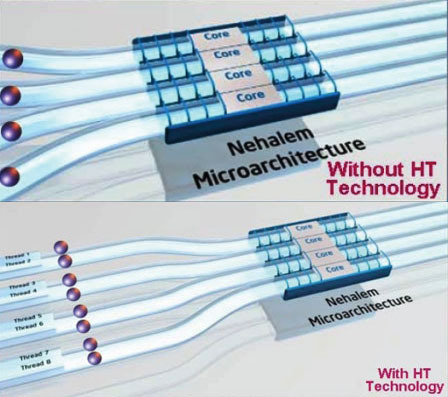


Figura . Hilos del procesador

**HILOS PRINCIPALES**

Los hilos principales o de interfaz de usuario se encargan de manejar la interacción con el usuario, como la actualización de la interfaz gráfica y la captura de eventos del teclado y el mouse. Los hilos principales son administrados por la aplicación y tienen prioridad sobre los hilos secundarios.

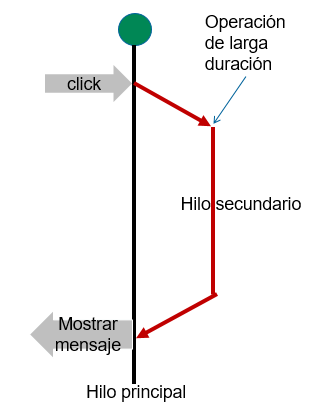


Figura . Hilos de Interfaz de Usuario

**HILOS SECUNDARIOS O DE TRABAJO**

Los hilos secundarios o de trabajo son utilizados para realizar tareas no críticas, como el procesamiento de datos o la descarga de archivos. Estos hilos son administrados por la aplicación y tienen menor prioridad que los hilos principales o del procesador. Los hilos secundarios son ideales para tareas que pueden ser ejecutadas de manera independiente y no afectan la interacción con el usuario.

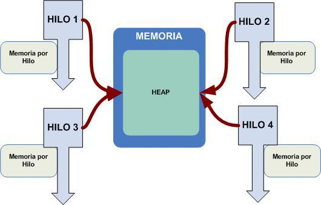


Figura . Hilos de trabajo

### **GESTIÓN DE HILOS**

La gestión de hilos es una técnica importante en programación que permite la ejecución concurrente de diferentes tareas dentro de un mismo proceso. Esto puede mejorar la eficiencia y el rendimiento de una aplicación, especialmente en sistemas con múltiples núcleos o procesadores.

La gestión de hilos también puede ser complicada, ya que los hilos comparten el mismo espacio de memoria y recursos, lo que puede llevar a conflictos y errores si no se gestionan adecuadamente (Davizhe, 2014). Por ejemplo, si dos hilos intentan acceder a la misma variable al mismo tiempo, puede haber errores de sincronización. Por lo tanto, es importante utilizar técnicas de sincronización y control de acceso para garantizar la correcta ejecución de los hilos.

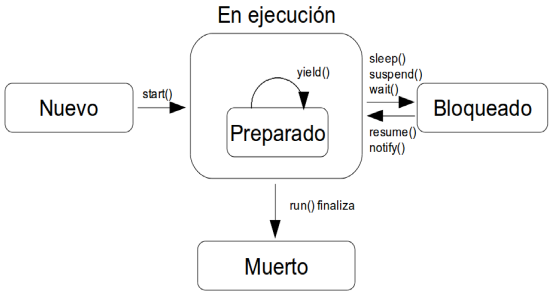


Figura . Gestión de Hilos

Durante el ciclo de vida de un Hilo, éste se puede encontrar en diferentes estados:

Tabla . Estados de los hilos

|  |  |
| --- | --- |
| **Estado** | **Descripción** |
| Ejecución | El hilo está siendo ejecutado. |
| Listo para ejecución | El hilo está en la lista de hilos listos para ser ejecutados, tan pronto disponga cada uno de tiempo de la CPU. |
| Suspensión | La ejecución del hilo es suspendida temporalmente. |
| Reanudación | Un hilo suspendido vuelve a ejecutarse continuando su tarea en el punto donde la dejó. |
| Bloqueado | Un hilo está en espera de un determinado recurso. |

### **INTERRUPCIONES**

La interrupción de hilos es una forma de indicar a un hilo que debe detenerse en la próxima oportunidad en la que pueda hacerlo de manera segura. En C#, se puede interrumpir un hilo llamando al método Interrupt de la clase Thread, lo que provocará la generación de una excepción ThreadInterruptedException en el hilo (Microsoft, 2022). Es importante tener en cuenta que la interrupción de hilos no garantiza que el hilo se detenga de inmediato, sino que proporciona una señal para que el hilo se detenga en la próxima oportunidad en la que pueda hacerlo de manera segura. Es responsabilidad del código del hilo detectar y manejar esta señal de manera adecuada.

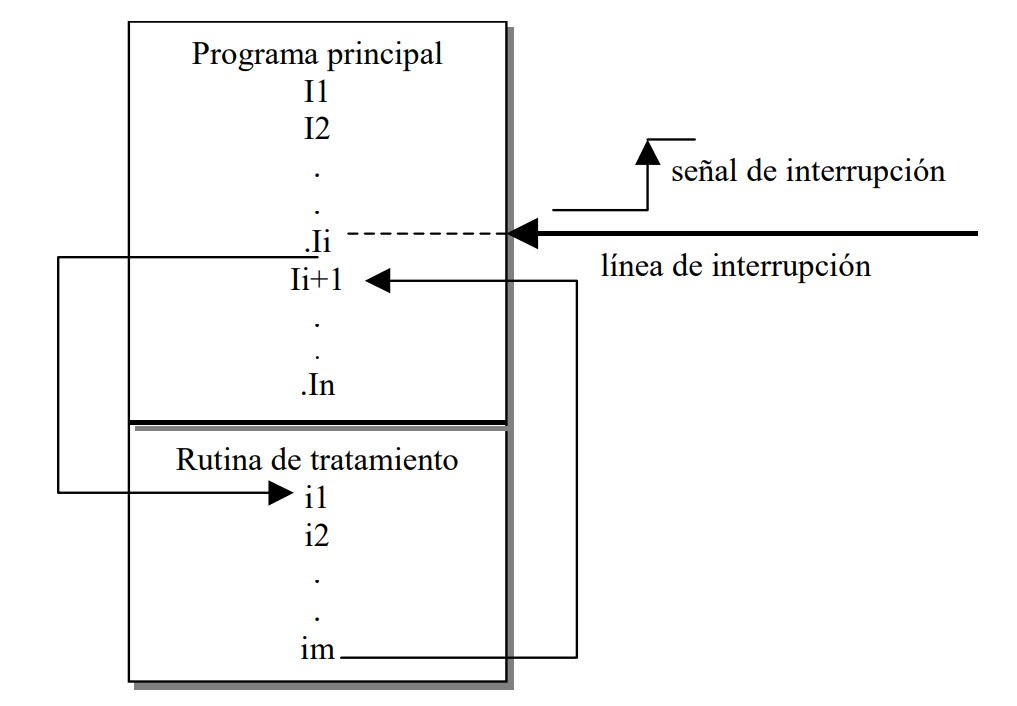


Figura . Interrupción de hilos

### **HILOS CON C#**

En C#, se pueden utilizar hilos para mejorar la eficiencia y el rendimiento de una aplicación al permitir que diferentes tareas se ejecuten al mismo tiempo en lugar de secuencialmente (Microsoft, 2022). Para utilizar hilos en C#, es necesario crear una clase que herede de la clase "Thread" y sobreescribir el método "Run()". Este método contiene el código que se ejecutará en el hilo. Una vez creada la clase, se puede instanciar y ejecutar mediante el método "Start()".

Los hilos también pueden tener prioridades, lo que permite controlar el orden en que se ejecutan en caso de que haya más de un hilo en ejecución al mismo tiempo. Además, es posible detener, pausar o reanudar la ejecución de un hilo en cualquier momento utilizando los métodos "Abort()", "Suspend()" y "Resume()", respectivamente.



Figura . Logo identificativo de C# con Hilos

**PROPIEDADES**

Tabla . Propiedades de la Clase Thread

|  |  |
| --- | --- |
| **Propiedad** | **Descripción** |
| ApartmentState | Obsoleto.  Obtiene o establece el estado de apartamento de este subproceso. |
| CurrentCulture | Obtiene o establece la referencia cultural del subproceso actual. |
| CurrentPrincipal | Obtiene o establece la entidad de seguridad actual del subproceso (de la seguridad basada en roles). |
| [CurrentThread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.currentthread?view=net-7.0#system-threading-thread-currentthread) | Obtiene el subproceso actualmente en ejecución. |
|  |  |
| [CurrentUICulture](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.currentuiculture?view=net-7.0#system-threading-thread-currentuiculture) | Obtiene o establece la referencia cultural actual utilizada por el administrador de recursos para buscar recursos específicos de la referencia cultural en tiempo de ejecución. |
| [ExecutionContext](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.executioncontext?view=net-7.0#system-threading-thread-executioncontext) | Obtiene un objeto [ExecutionContext](https://learn.microsoft.com/es-es/dotnet/api/system.threading.executioncontext?view=net-7.0) que contiene información sobre los distintos contextos del subproceso actual. |
| [IsAlive](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isalive?view=net-7.0#system-threading-thread-isalive) | Obtiene un valor que indica el estado de ejecución del subproceso actual. |
| [IsBackground](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isbackground?view=net-7.0#system-threading-thread-isbackground) | Obtiene o establece un valor que indica si un subproceso es o no un subproceso en segundo plano. |
| [IsThreadPoolThread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isthreadpoolthread?view=net-7.0#system-threading-thread-isthreadpoolthread) | Obtiene un valor que indica si un subproceso pertenece al grupo de subprocesos administrados o no. |
| [ManagedThreadId](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.managedthreadid?view=net-7.0#system-threading-thread-managedthreadid) | Obtiene un identificador único para el actual subproceso administrado. |
| [Name](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.name?view=net-7.0#system-threading-thread-name) | Obtiene o establece el nombre del subproceso. |
| [Priority](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.priority?view=net-7.0#system-threading-thread-priority) | Obtiene o establece un valor que indica la prioridad de programación de un subproceso. |
| [ThreadState](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.threadstate?view=net-7.0#system-threading-thread-threadstate) | Obtiene un valor que contiene los estados del subproceso actual. |

**MÉTODOS**

Tabla . Métodos de la Clase Thread

|  |  |
| --- | --- |
| **Métodos** | **Descripción** |
| [Abort()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0#system-threading-thread-abort) | **Obsoleto.**  Produce una excepción [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception?view=net-7.0) en el subproceso en el que se invoca, para iniciar el proceso de finalización del subproceso. Normalmente, una llamada a este método finaliza el subproceso. |
| [Abort(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0#system-threading-thread-abort(system-object)) | **Obsoleto.**  Produce una excepción [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception?view=net-7.0) en el subproceso en el que se invoca, para iniciar el proceso de finalización del subproceso, proporcionando al mismo tiempo información sobre excepciones relativa a la terminación del subproceso. Normalmente, una llamada a este método finaliza el subproceso. |
| [Equals(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.object.equals?view=net-7.0#system-object-equals(system-object)) | Determina si el objeto especificado es igual que el objeto actual.  (Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) |
| [Finalize()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.finalize?view=net-7.0#system-threading-thread-finalize) | Se asegura de que los recursos se liberan y que se llevan a cabo otras operaciones de limpieza cuando el recolector de elementos no utilizados recupere el objeto [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0). |
| [Interrupt()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt?view=net-7.0#system-threading-thread-interrupt) | Interrumpe un subproceso que se encuentra en estado de subproceso [WaitSleepJoin](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-waitsleepjoin). |
| [Join()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join?view=net-7.0#system-threading-thread-join) | Bloquea el subproceso de llamada hasta que el subproceso representado por esta instancia finaliza, pero continúa bombeando SendMessage y COM estándar. |
| [Join(Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join?view=net-7.0#system-threading-thread-join(system-int32)) | Bloquea el subproceso de llamada hasta que el subproceso representado por esta instancia finaliza o transcurre el tiempo especificado, pero continúa bombeando SendMessage y COM estándar. |
| [Join(TimeSpan)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join?view=net-7.0#system-threading-thread-join(system-timespan)) | Bloquea el subproceso de llamada hasta que el subproceso representado por esta instancia finaliza o transcurre el tiempo especificado, pero continúa bombeando SendMessage y COM estándar. |
| [MemberwiseClone()](https://learn.microsoft.com/es-es/dotnet/api/system.object.memberwiseclone?view=net-7.0#system-object-memberwiseclone) | Crea una copia superficial del [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0) actual.  (Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) |
| [MemoryBarrier()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.memorybarrier?view=net-7.0#system-threading-thread-memorybarrier) | Sincroniza el acceso a la memoria de la siguiente forma: el procesador que ejecuta el subproceso actual no puede reordenar las instrucciones de forma que los accesos a la memoria anteriores a la llamada a [MemoryBarrier()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.memorybarrier?view=net-7.0" \l "system-threading-thread-memorybarrier) se ejecuten después de los accesos a memoria que siguen a la llamada a [MemoryBarrier()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.memorybarrier?view=net-7.0" \l "system-threading-thread-memorybarrier). |
| [ResetAbort()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.resetabort?view=net-7.0#system-threading-thread-resetabort) | **Obsoleto.**  Cancela un método [Abort(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0" \l "system-threading-thread-abort(system-object)) solicitado para el subproceso actual. |
| [Resume()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.resume?view=net-7.0#system-threading-thread-resume) | **Obsoleto.**  Reanuda un subproceso que se ha suspendido. |
| [Sleep(Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep?view=net-7.0#system-threading-thread-sleep(system-int32)) | Suspende el subproceso actual durante el número de milisegundos especificado. |
| [Sleep(TimeSpan)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep?view=net-7.0#system-threading-thread-sleep(system-timespan)) | Suspende el subproceso actual durante la cantidad de tiempo especificada. |
| [SpinWait(Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.spinwait?view=net-7.0#system-threading-thread-spinwait(system-int32)) | Hace que un subproceso espere el número de veces definido por el parámetro iterations. |
| [Start()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0#system-threading-thread-start) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running). |
| [Start(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0#system-threading-thread-start(system-object)) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running); también puede proporcionar un objeto que contiene datos para que los use el método ejecutado por el subproceso. |
| [Suspend()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.suspend?view=net-7.0#system-threading-thread-suspend) | **Obsoleto.**  Suspende el subproceso o, si este ya se ha suspendido, no tiene efecto alguno. |
| [ToString()](https://learn.microsoft.com/es-es/dotnet/api/system.object.tostring?view=net-7.0#system-object-tostring) | Devuelve una cadena que representa el objeto actual.  (Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) |
| [UnsafeStart()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.unsafestart?view=net-7.0#system-threading-thread-unsafestart) | Hace que el sistema operativo cambie el estado de la instancia actual a [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running). |
| [Yield()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.yield?view=net-7.0#system-threading-thread-yield) | Hace que el subproceso que realiza la llamada ceda la ejecución a otro subproceso que está listo para ejecutarse en el procesador actual. El sistema operativo selecciona el subproceso al que se va a ceder la ejecución. |

**MÉTODO JOIN Y START**

El método Join() permite que un hilo espere a que otro hilo termine su ejecución. Al llamar a Join() en un hilo, el hilo actual se bloqueará hasta que el hilo al que se le llamó Join() termine su ejecución. Por ejemplo, si tienes dos hilos A y B, y al llamar a Join() en el hilo A para unirse al hilo B, el hilo A se bloqueará hasta que el hilo B termine su ejecución.

El método Start() es el encargado de iniciar la ejecución de un hilo. Al tener un hilo A creado pero no iniciado, llamar a A.Start() comenzará a ejecutar el hilo A. Es importante tener en cuenta que el método Start() solo puede ser llamado una vez en un hilo. Al llamar a Start() en un hilo que ya ha sido iniciado anteriormente, se lanzará una excepción ThreadStateException.

# PARTE PRÁCTICA

Para esta práctica se implementará un programa por terminal que use dos hilos, el primer hilo debe ejecutar y finalizar su proceso para que el segundo comience con la ejecución, para esto se debe hacer uso del método Join() de la clase Thread de C#.

## CREACIÓN DEL PROYECTO

1. Para comenzar con la creación del proyecto, abrir Visual Studio, seleccionar y dar clic en la opción Create a new project.

Graphical user interface, text, application

Description automatically generated

Figura . Ventana de proyectos Visual Studio

1. En la siguiente ventana, seleccionar la plantilla del proyecto Console App y clic en Next.

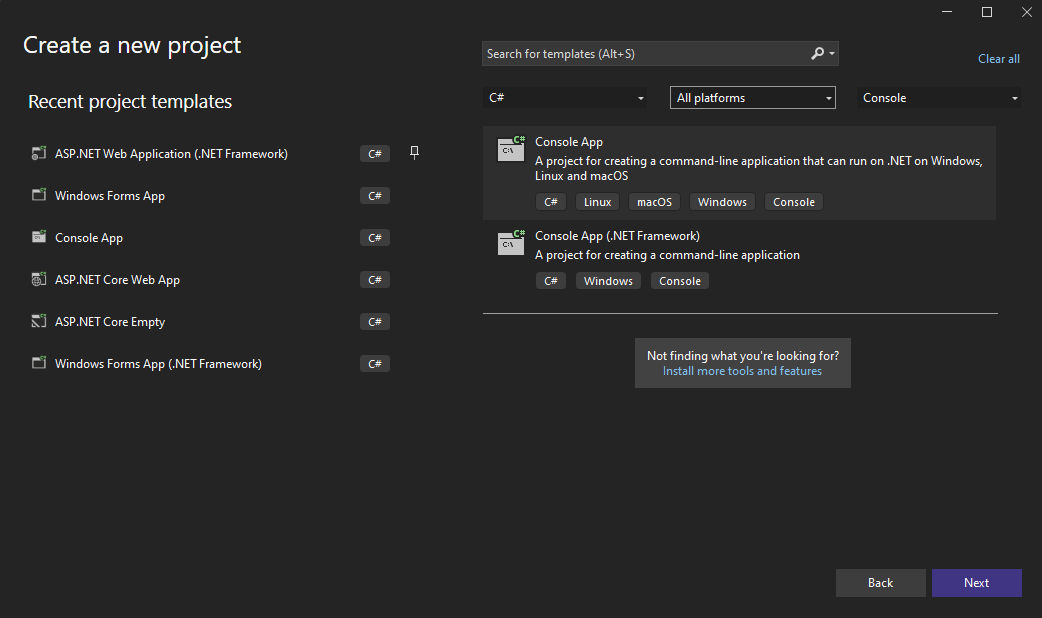


Figura . Plantillas de proyectos Visual Studio

1. A continuación, ingresar el nombre del proyecto, la ubicación para almacenar los archivos generados y clic en Next.

Graphical user interface, application

Description automatically generated

Figura . Ingreso del nombre y carpeta para el programa

1. Luego seleccionar la version del Framework .NET y clic en Create.

Graphical user interface, application, Teams

Description automatically generated

Figura . Selección de la version del Framework .NET

1. Una vez la herramienta Visual Studio finalice con la configuración y creación del proyecto, se muestra la clase principal Program.cs y una lista de archivos adicionales necesarios para el programa.

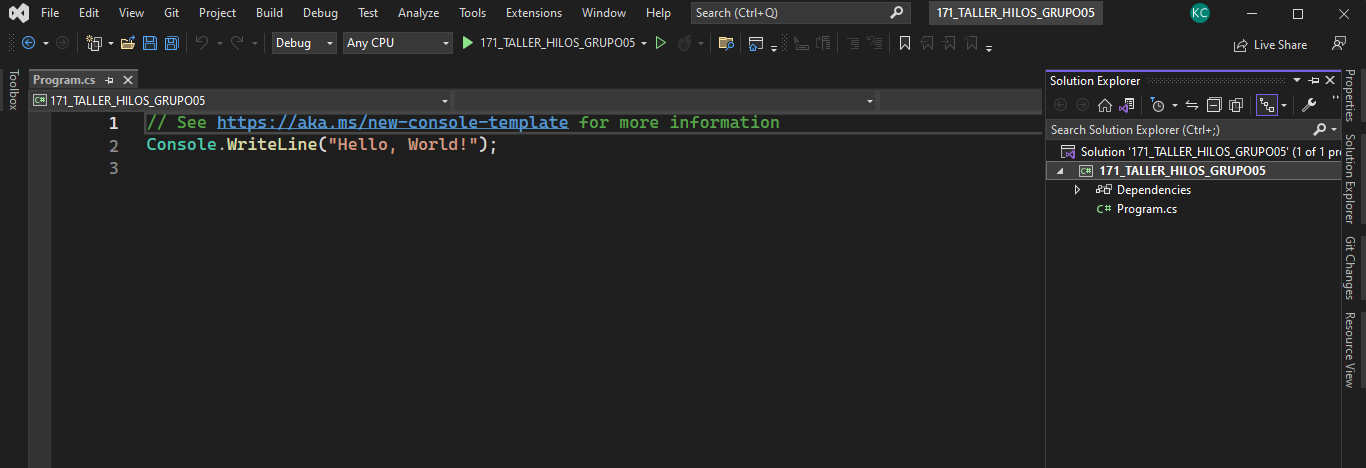


Figura . Archivos generados para el programa

## ARQUITECTURA MVC PARA EL PROYECTO

Para definir la arquitectura MVC (Modelo-Vista-Controlador), se crean tres carpetas para organizar y clasificar las clases del proyecto.

1. Para crear una carpeta, dirigirse a la sección Solution Explorer y dar clic derecho en el segundo item de la lista, después dirigirse a la opción Add y dar clic en New Folder. Como se requieren tres carpetas, repetir estre proceso tres veces.

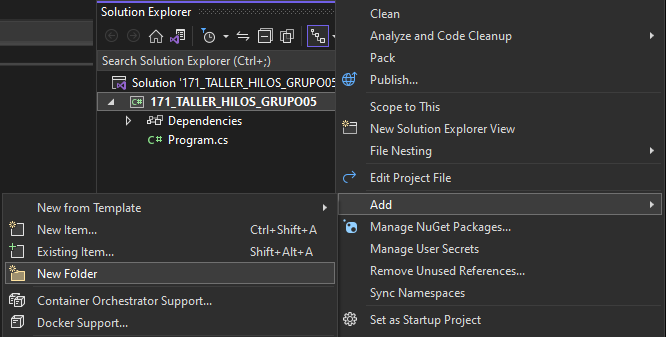


Figura . Creación de carpetas para MVC

1. Una vez creadas las carpetas dar clic sobre cada una y presionar la tecla F2 para renombrarlas con los nombres modelo, vista y controlador.

Text

Description automatically generated

Figura . Arquitectura MVC para el programa

## PRÁCTICA VIDEO 171

Para esta parte de la práctica se crearán dos hilos, estos imprimirán un mensaje que contendrá el nombre del hilo cinco veces, usando el método Join(), el primer hilo debe ejecutar su proceso hasta terminarlo para que el segundo ejecute su proceso y lo termine, cuando los dos hilos terminen de ejecutar sus procesos se imprimirá un mensaje final.

### **CODIFICACIÓN** **CONTROLADOR**

1. Para añadir una nueva clase, dirigirse a la sección Solution Explorer, dar clic derecho sobre la carpeta modelo, luego Add y clic en la opción Class.

A screenshot of a computer

Description automatically generated with medium confidence

Figura . Creación de Clase en el controlador

1. En la siguiente ventana, ingresar el nombre de la clase como HiloControlador y clic en Add.

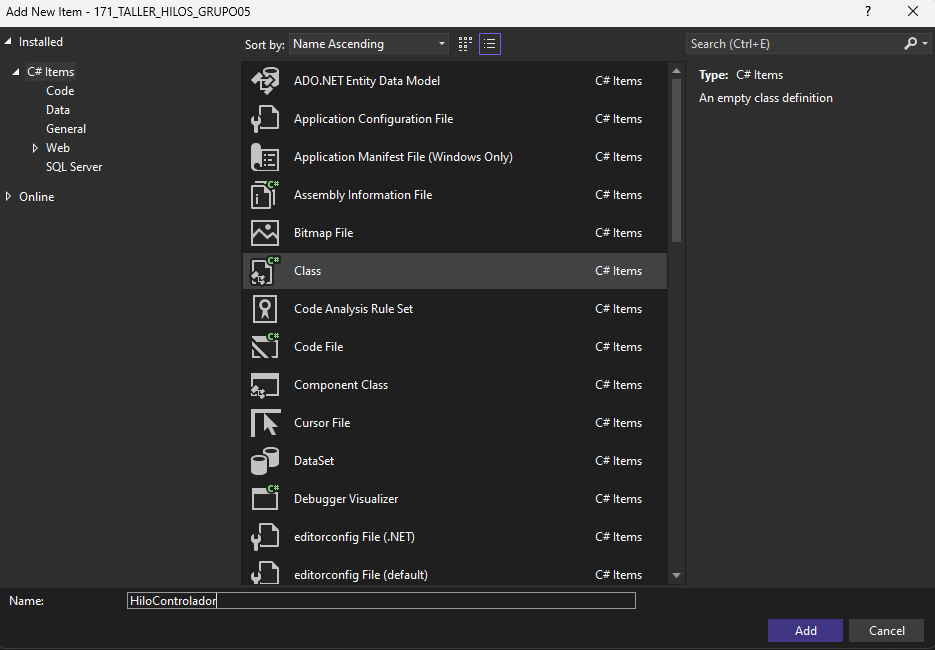


Figura . Asignación del nombre HiloControlador para la Clase

1. La clase HiloControlador tendrá una propiedad hilo, este objeto tendrá el método que imprimirá el mensaje cinco veces con el nombre del hilo que se está ejecutando.

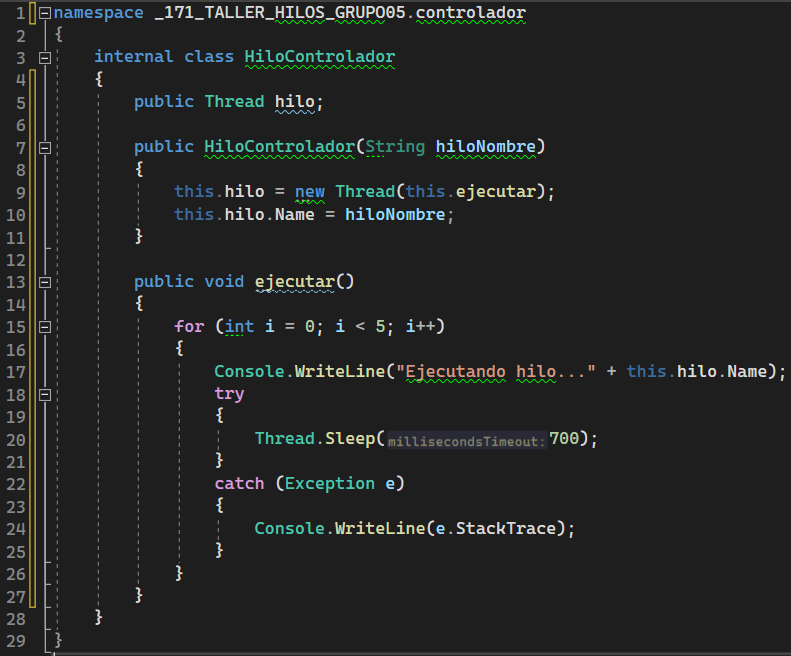


Figura . Codificación de la Clase HiloControlador

Tabla . Codificación del archivo HiloControlador.cs práctica video 171

namespace \_171\_TALLER\_HILOS\_GRUPO05.controlador

{

internal class HiloControlador

{

public Thread hilo;

public HiloControlador(String hiloNombre)

{

this.hilo = new Thread(this.ejecutar);

this.hilo.Name = hiloNombre;

}

public void ejecutar()

{

for (int i = 0; i < 5; i++)

{

Console.WriteLine("Ejecutando hilo..." + this.hilo.Name);

try

{

Thread.Sleep(700);

}

catch (Exception e)

{

Console.WriteLine(e.StackTrace);

}

}

}

}

}

1. Para gestionar los dos hilos se crea una nueva clase en el controlador, en el campo nombre se ingresará GestionHilo y clic en Add.

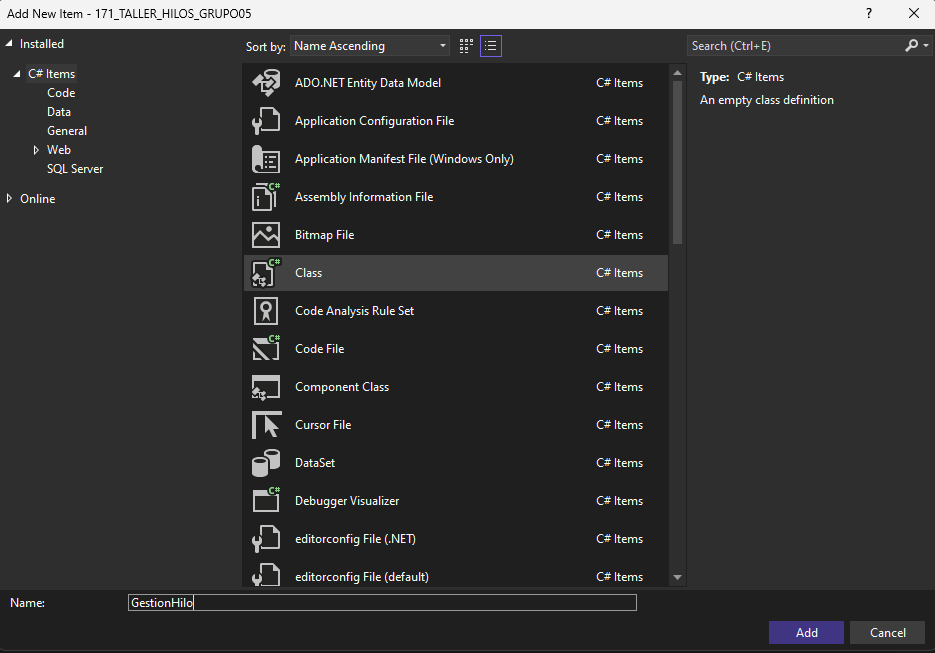


Figura . Creación de la Clase GestionHilo

1. La clase GestionHilo tendrá una función, esta será la encargada de instanciar los dos hilos con el método que imprime el mensaje cinco veces. El método Start() inicia la ejecución del hilo 1 y con el método Join() obligara a que este termine su ejecucion para que el hilo 2 se ejecute.

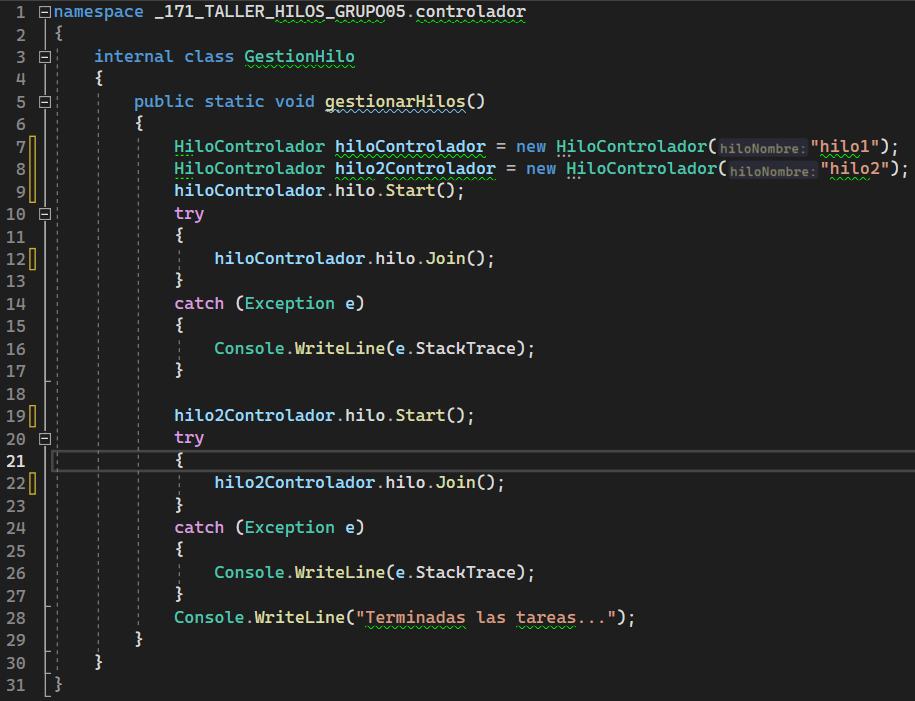


Figura . Codificación de la clase GestionHilo

Tabla . Codificación del archivo GestionHilo.cs práctica video 171

namespace \_171\_TALLER\_HILOS\_GRUPO05.controlador

{

internal class GestionHilo

{

public static void gestionarHilos()

{

HiloControlador hiloControlador = new HiloControlador("hilo1");

HiloControlador hilo2Controlador = new HiloControlador("hilo2");

hiloControlador.hilo.Start();

try

{

hiloControlador.hilo.Join();

}

catch (Exception e)

{

Console.WriteLine(e.StackTrace);

}

hilo2Controlador.hilo.Start();

try

{

hilo2Controlador.hilo.Join();

}

catch (Exception e)

{

Console.WriteLine(e.StackTrace);

}

Console.WriteLine("Terminadas las tareas...");

}

}

}

1. Por último, hay que llamar el método gestionarHilos() de la clase GestionHilo.cs para que el programa se ejecute en la clase Program.cs.

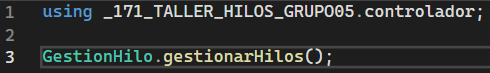


Figura . Codificación de la Clase Program

Tabla . Codificación del archivo Program.cs práctica video 171

using \_171\_TALLER\_HILOS\_GRUPO05.controlador;

GestionHilo.gestionarHilos();

## PRÁCTICA VIDEO 172

Para esta parte de la práctica se hará uso del método Join() dentro de un hilo externo que se haya creado, deteniendo la ejecución del hilo creado, hasta que un hilo interno finalice la ejecución de su proceso, de esta forma se debe obtener la misma salida de ejecución que en la práctica anterior.

### **CODIFICACIÓN CONTROLADOR**

1. En el controlador se creará una nueva clase con el nombre Hilo2Controlador y luego clic en Add, este será el hilo externo que contendrá el hilo interno al cual se le encadenara el método Join() para que este finalice su proceso de ejecución.

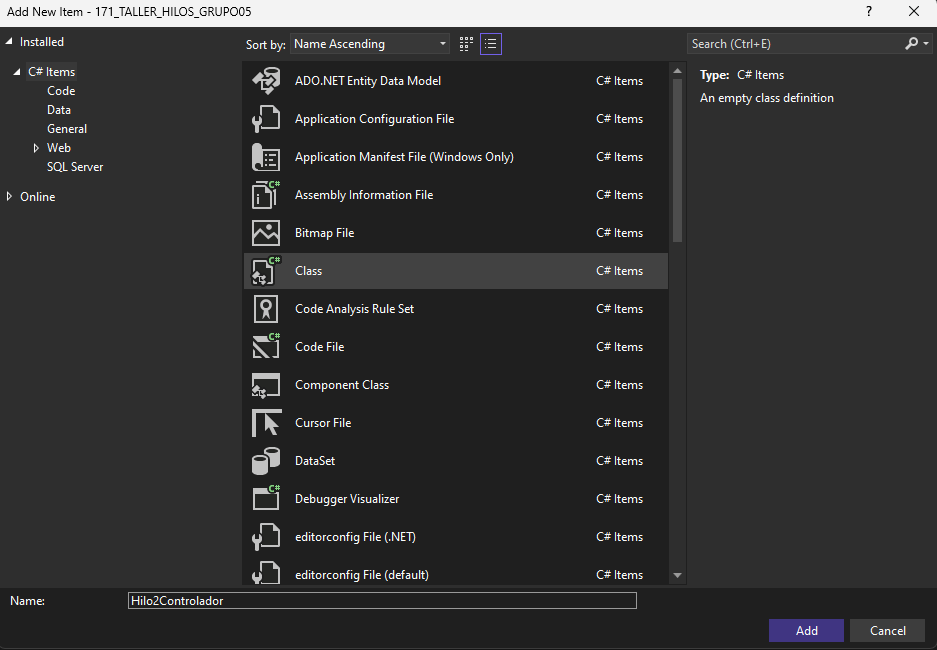


Figura . Creación de la clase Hilo2Controlador

1. La clase Hilo2Controlador recibirá como parámetro al hilo interno y se le encadenara al método Join() en la función que se ejecutara en el hilo externo.

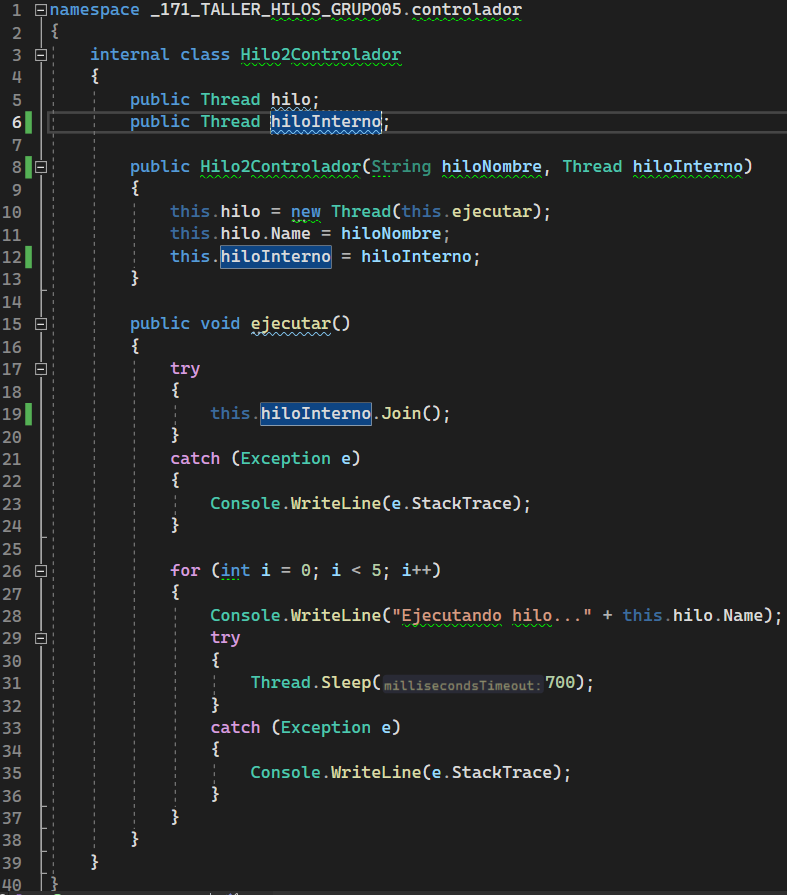


Figura . Codificación de la Clase Hilo2Controlador

Tabla . Codificación del archivo Hilo2Controlador.cs práctica video 172

namespace \_171\_TALLER\_HILOS\_GRUPO05.controlador{

internal class Hilo2Controlador{

public Thread hilo;

public Thread hiloInterno;

public Hilo2Controlador(String hiloNombre, Thread hiloInterno){

this.hilo = new Thread(this.ejecutar);

this.hilo.Name = hiloNombre;

this.hiloInterno = hiloInterno;

}

public void ejecutar(){

try{

this.hiloInterno.Join();

}

catch (Exception e){

Console.WriteLine(e.StackTrace);

}

for (int i = 0; i < 5; i++)

{

Console.WriteLine("Ejecutando hilo..." + this.hilo.Name);

try

{

Thread.Sleep(700);

}

catch (Exception e)

{

Console.WriteLine(e.StackTrace);

}

}

}

}

}

1. En el método gestionarHilos() de la clase GestionHilo se debe pasar como parámetro el objeto de la clase HiloControlador al objeto de la clase Hilo2Controlador, luego ejecutar el método Start() en el hilo2Controlador (externo) y luego al hiloControlador (interno).

Text

Description automatically generated

Figura . Codificación de la Clase GestionHilo

Tabla . Codificación del archivo GestionHilo.cs práctica video 172

namespace \_171\_TALLER\_HILOS\_GRUPO05.controlador

{

internal class GestionHilo

{

public static void gestionarHilos()

{

HiloControlador hiloControlador = new HiloControlador("hilo1");

Hilo2Controlador hilo2Controlador = new Hilo2Controlador("hilo2", hiloControlador.hilo);

hilo2Controlador.hilo.Start();

hiloControlador.hilo.Start();

}

}

}

## ESTRUCTURA DE LA APLICACIÓN

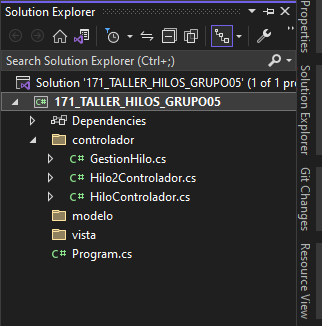


Figura . Estructura final del programa con MVC

## EJECUCIÓN DEL PROYECTO

Para ejecutar el proyecto dirigirse a la barra de herramientas superior y dar clic en Start Without Debugging o también pulsando las teclas Ctrl+F5.

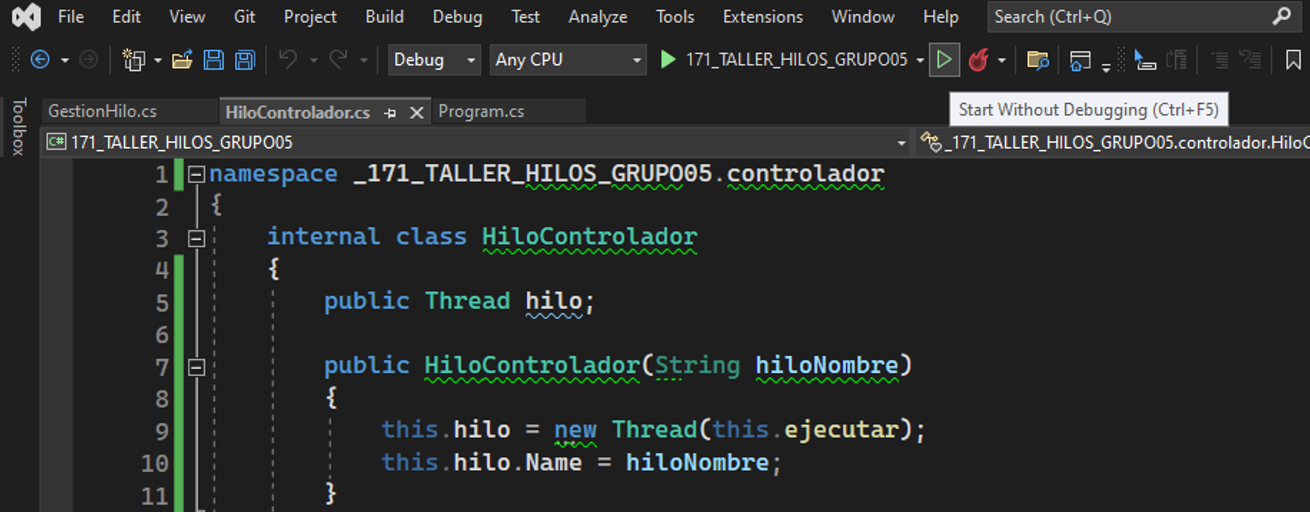


Figura . Ejecución del programa

### **EJECUCIÓN VIDEO 171**

Una vez se ejecute el proyecto, el hilo 1 será el primero que ejecute y termine su proceso de impresiones por consola, esto debido a que tiene encadenado el método Join() y que el hilo 2 tiene encadenado el método Start() y Join() después del hilo 1. Lo último que se mostrará por consola, será un mensaje para mostrar que los procesos de los hilos terminaron.

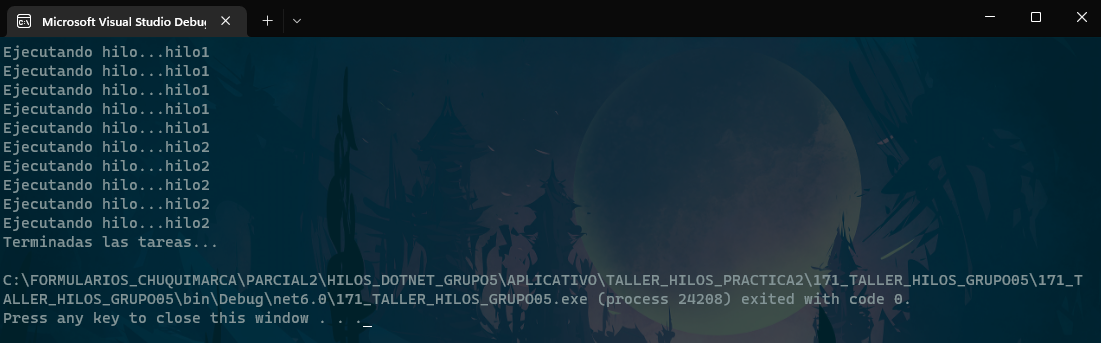


Figura . Salida de la ejecución del video 171

### **EJECUCIÓN VIDEO 172**

La salida será la misma, pero ahora el método Join() hará que el hilo interno detenga la ejecución del hilo externo hasta que este finalice la ejecución de su proceso. Por último, se imprime el mensaje que la ejecución de los procesos de los hilos termino.

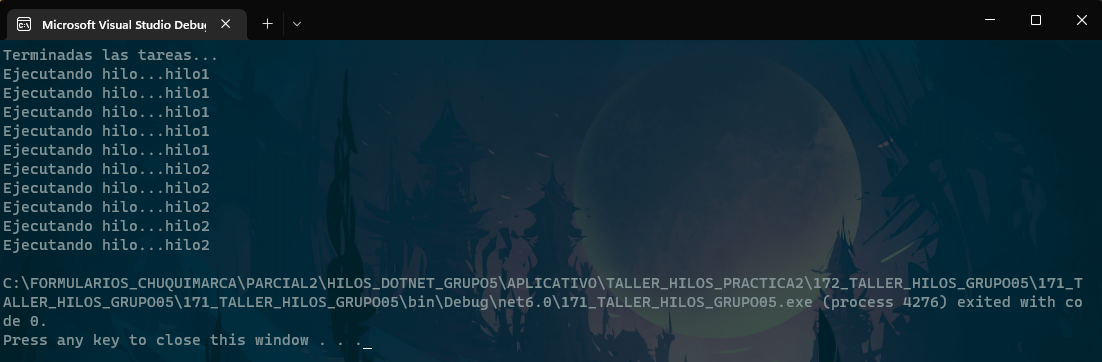


Figura . Salida de la ejecución del video 172

# CONCLUSIONES

* El método Join() de la clase Thread permite que un hilo espere a que otro hilo termine su ejecución antes de comenzar. De esta manera, se puede controlar el orden de ejecución de los hilos para evitar problemas de flujo de información y resultados inconsistentes.
* El método Join() proporciona una forma para controlar la ejecución concurrente de hilos y asegurar que se ejecuten en un orden específico, puede usarse para evitar problemas de concurrencia y sincronización de hilos.
* El método Join() puede usarse para asegurar que un hilo termine su ejecución antes de que otro hilo comience a ejecutarse, lo que puede ser útil para realizar tareas de limpieza o inicialización, es una opción más segura que interrumpir un hilo directamente, ya que permite que el hilo termine su ejecución de manera controlada y consistente.

# RECOMENDACIONES

* Utilizar el método Join() para sincronizar la ejecución de hilos y asegurar que se ejecuten en el orden deseado, y en lugar de interrumpir hilos directamente para permitir que los hilos terminen su ejecución de manera controlada y consistente.
* Considerar la posibilidad de usar otras opciones de sincronización de hilos, como Sleep(), Wait() y Pulse(), junto con el método Join() para controlar la ejecución de hilos de manera más precisa.
* Manejar adecuadamente las excepciones ThreadInterruptedException que pueden lanzar el método Join() si el hilo que está esperando es interrumpido.

# REFERENCIAS

* Blancarte, O. (29 de Marzo de 2017). *Concurrencia vs Paralelismo*. Obtenido de oscarblancarteblog: https://www.oscarblancarteblog.com/2017/03/29/concurrencia-vs-paralelismo/
* Davizhe. (7 de Octubre de 2014). *GESTIÓN DE HILOS*. Obtenido de davidizquierdoheras: https://davidizquierdoheras.wordpress.com/2014/10/07/gestion-de-hilos/
* Elorduy, G. (2020). *Introducción a los hilos*. Obtenido de open source initiative: https://javaparajavatos.wordpress.com/2017/05/06/introduccion-a-los-hilos/
* Microsoft. (22 de Septiembre de 2022). *Paseo por el lenguaje C#*. Obtenido de Learn Microsoft: https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/
* Milagros, B. (20 de Marzo de 2022). *¿Qué es Visual Studio? ¡El desarrollo de software nunca fue más fácil!* Obtenido de crehana: https://www.crehana.com/blog/transformacion-digital/que-es-visual-studio/
* Nakayama, A. (2009). Guía práctica de estudio 12: Hilos. México.
* Pellicer, P. (21 de Octubre de 2021). *¿Qué es el .NET? ¿Para qué sirve?* Obtenido de emagister: https://www.emagister.com/blog/que-es-el-net-para-que-sirve/
* Rossainz, M. L. (2015). *Programación Concurrente y Paralela.* Puebla.