

**Tema**

HILOS SINCRONIZACIÓN PROYECTO DOS DOTNET

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

06/08/2023

HILOS SINCRONIZACIÓN PROYECTO DOS DOTNET

[1. MARCO TEÓRICO 6](#_Toc124900951)

[1.1. INTRODUCCIÓN 6](#_Toc124900952)

[1.2. OBJETIVOS 6](#_Toc124900953)

[**1.2.1.** **OBJETIVO GENERAL** 6](#_Toc124900954)

[**1.2.2.** **OBJETIVOS ESPECÍFICOS** 6](#_Toc124900955)

[1.3. HILOS 6](#_Toc124900956)

[**1.3.1. CLASE SYSTEM.THREADING** 7](#_Toc124900957)

[**1.3.2. THREADSTART** 12](#_Toc124900958)

[**1.3.3. THREADPOOL** 12](#_Toc124900959)

[**1.3.4. THREADSTATE** 15](#_Toc124900960)

[**1.3.5. THREADPRIORITY** 18](#_Toc124900961)

[**1.3.6. THREADLOCAL<T>** 18](#_Toc124900962)

[**1.3.7. VERSIONAMIENTO** 20](#_Toc124900963)

[**1.3.8. CREACÓN DE SUBPROCESOS EN DOTNET** 20](#_Toc124900964)

[**1.3.9. DESTRUCCIÓN DE SUBPROCESOS EN DOTNET** 21](#_Toc124900965)

[**1.3.10. VENTAJAS Y DESVENTAJAS DE LOS HILOS** 21](#_Toc124900966)

[**1.3.11. CICLO DE VIDA DE UN HILO** 22](#_Toc124900967)

[**1.3.12. PROPIEDADES Y MÉTODOS DE LA CLASE THREAD** 22](#_Toc124900968)

[**1.3.13. SINCRONIZACIÓN DE HILOS** 31](#_Toc124900969)

[**1.3.14. EXCEPCIONES EN HILOS** 33](#_Toc124900970)

[**1.3.15. INTERRUPCIONES Y BLOQUEOS** 34](#_Toc124900971)

[**1.3.16. CONSTRUCTORES** 35](#_Toc124900972)

[1.4. FRAMEWORK 36](#_Toc124900973)

[1.5. SISTEMAS MONOTAREA Y MULTITAREA 36](#_Toc124900974)

[1.6. PATRONES DE DISEÑO 36](#_Toc124900975)

[**1.6.1. MODELO – VISTA – CONTROLADOR (MVC)** 37](#_Toc124900976)

[1.7. NOMENCLATURA DE PROGRAMACIÓN 37](#_Toc124900977)

[2. PARTE PRÁCTICA 37](#_Toc124900978)

[2.1. CREACIÓN DE PRIMER PROYECTO DOTNET 37](#_Toc124900979)

[2.2. CREACIÓN DE PAQUETES EN PRIMER PROYECTO DOTNET 40](#_Toc124900980)

[2.3. CREACIÓN DE ARCHIVOS .CS EN PRIMER PROYECTO DOTNET 41](#_Toc124900981)

[**2.3.1 CODIFICACIÓN DEL ARCHIVO HILO.CS** 41](#_Toc124900982)

[**2.3.2 CODIFICACIÓN DEL ARCHIVO HILOCONTROLADOR.CS** 44](#_Toc124900983)

[**2.3.3 CODIFICACIÓN DEL ARCHIVO PRINCIPAL.CS** 46](#_Toc124900984)

[2.4. CREACIÓN DE SEGUNDO PROYECTO DOTNET 48](#_Toc124900985)

[2.5. CREACIÓN DE PAQUETES EN SEGUNDO PROYECTO DOTNET 51](#_Toc124900986)

[2.6. CREACIÓN DE ARCHIVOS .CS EN SEGUNDO PROYECTO DOTNET 52](#_Toc124900987)

[**2.6.1 CODIFICACIÓN DEL ARCHIVO HILO.CS** 52](#_Toc124900988)

[**2.6.2 CODIFICACIÓN DEL ARCHIVO HILOCONTROLADOR.CS** 54](#_Toc124900989)

[**2.6.3 CODIFICACIÓN DEL ARCHIVO PRINCIPAL.CS** 57](#_Toc124900990)

[2.7 FUNCIONALIDAD 59](#_Toc124900991)

[**2.7.1** **ESTRUCTURA DE LAS APLICACIÓNES** 59](#_Toc124900992)

[2.8 DESEMPEÑO DE LAS APLICACIÓNES 59](#_Toc124900993)

[**2.8.1 PRIMER PROYECTO DOTNET HILOSVIDEO171\_DOTNET\_GRUPO1** 59](#_Toc124900994)

[**2.8.2 SEGUNDO PROYECTO DOTNET HILOSVIDEO172\_DOTNET\_GRUPO1** 60](#_Toc124900995)

[3. CONCLUSIONES 61](#_Toc124900996)

[4. RECOMENDACIONES 61](#_Toc124900997)

[5. REFERENCIAS 62](#_Toc124900998)

**INDICE DE IMÁGENES**

[Figura 1. Ejemplo de la sintaxis de hilos 7](#_Toc124900866)

[Figura 2. Uso de Paralelismo en clase System.Threading 7](#_Toc124900867)

[Figura 3. Recorrido de ThreadStart 12](#_Toc124900868)

[Figura 4. Tareas de ThreadPool 13](#_Toc124900869)

[Figura 5. Ejecución a partir de MAIN THREAD 17](#_Toc124900870)

[Figura 6. Versionamiento Requerido para Threads de Dotnet 20](#_Toc124900871)

[Figura 7. Creación Subprocesos Dotnet 20](#_Toc124900872)

[Figura 8. Llamada a callback desde TokenThread 21](#_Toc124900873)

[Figura 9. Ciclo de vida de un Hilo 22](#_Toc124900874)

[Figura 10. Ejemplo Monitor.Exit C# 32](#_Toc124900875)

[Figura 11. Sincronización de Hilos C# 33](#_Toc124900876)

[Figura 12. Ejemplo de bloqueo con hilos en C# 34](#_Toc124900877)

[Figura 13.Interrupciones y Bloqueos 35](#_Toc124900878)

[Figura 14. Representación de ejecución de 4 procesos 36](#_Toc124900879)

[Figura 15. Ejemplos de las nomenclaturas de programación 37](#_Toc124900880)

[Figura 16. Apertura del IDE Visual Studio 2022 38](#_Toc124900881)

[Figura 17. Ayudante Herramientas del IDE Visual Studio 2022 38](#_Toc124900882)

[Figura 18. Creación de aplicación tipo consola IDE Visual Studio 2022 39](#_Toc124900883)

[Figura 19. Nombramiento de proyecto IDE Visual Studio 2022 39](#_Toc124900884)

[Figura 20. Selección de Framework IDE Visual Studio 2022 40](#_Toc124900885)

[Figura 21. Primera Vista IDE Visual Studio 2022 40](#_Toc124900886)

[Figura 22. Agregación de una nueva carpeta proyecto Dotnet 40](#_Toc124900887)

[Figura 23. Creación de paquetes MVC en Dotnet 41](#_Toc124900888)

[Figura 24. Agregación de nueva clase Dotnet 41](#_Toc124900889)

[Figura 25. Nombramiento de clase Hilo.cs Dotnet 42](#_Toc124900890)

[Figura 26. Creación del proyecto Hilo.cs Dotnet 42](#_Toc124900891)

[Figura 27. Codificación del archivo Hilo.cs 42](#_Toc124900892)

[Figura 28. Agregación de nueva clase Dotnet 44](#_Toc124900893)

[Figura 29. Nombramiento de la clase HiloControlador.cs 44](#_Toc124900894)

[Figura 30. Apertura básica de HiloControlador.cs 44](#_Toc124900895)

[Figura 31. Codificación del archivo HiloControlador.cs 45](#_Toc124900896)

[Figura 32. Creación de nueva clase Dotnet 46](#_Toc124900897)

[Figura 33. Selección y nombramiento de componente clase Dotnet 46](#_Toc124900898)

[Figura 34. Codificación del archivo Principal.cs 46](#_Toc124900899)

[Figura 35. Eliminación del archivo Program.cs 47](#_Toc124900900)

[Figura 36. Codificación del archivo Principal.cs 47](#_Toc124900901)

[Figura 37. Apertura del IDE Visual Studio 2022 48](#_Toc124900902)

[Figura 38. Ayudante Herramientas del IDE Visual Studio 2022 49](#_Toc124900903)

[Figura 39. Creación de aplicación tipo consola IDE Visual Studio 2022 49](#_Toc124900904)

[Figura 40. Nombramiento del proyecto Dotnet 50](#_Toc124900905)

[Figura 41. Selección de Framework IDE Visual Studio 2022 50](#_Toc124900906)

[Figura 42. Generación proyecto base Dotnet 51](#_Toc124900907)

[Figura 43. Agregación de paquetes en el proyecto Dotnet 51](#_Toc124900908)

[Figura 44. Agregación de paquetes MVC en Dotnet 51](#_Toc124900909)

[Figura 45. Agregación de nueva clase Dotnet 52](#_Toc124900910)

[Figura 46. Nombramiento de clase Hilo.cs Dotnet 52](#_Toc124900911)

[Figura 47. Creación del proyecto Hilo.cs Dotnet 53](#_Toc124900912)

[Figura 48. Codificación del archivo Hilo.cs 53](#_Toc124900913)

[Figura 49. Agregación de nueva clase Dotnet 54](#_Toc124900914)

[Figura 50. Nombramiento de la clase HiloControlador.cs 55](#_Toc124900915)

[Figura 51. Generación de HiloControlador.cs 55](#_Toc124900916)

[Figura 52. Codificación del archivo HiloControlador.cs 56](#_Toc124900917)

[Figura 53. Creación de nueva clase Dotnet 57](#_Toc124900918)

[Figura 54. Selección y nombramiento de componente clase Dotnet 57](#_Toc124900919)

[Figura 55. Codificación del archivo Principal.cs 57](#_Toc124900920)

[Figura 56. Eliminación del archivo Program.cs 58](#_Toc124900921)

[Figura 57. Codificación exitosa Principal.cs 58](#_Toc124900922)

[Figura 58. Estructura del primer proyecto Dotnet HilosVideo171\_Dotnet\_Grupo1 59](#_Toc124900923)

[Figura 59. Estructura del segundo proyecto Dotnet HilosVideo172\_Dotnet\_Grupo1 59](#_Toc124900924)

[Figura 60. Ejecución de HilosVideo171\_Dotnet\_Grupo1 60](#_Toc124900925)

[Figura 61. Ejecución en consola de HilosVideo171\_Dotnet\_Grupo1 60](#_Toc124900926)

[Figura 62. Botón para ejecución HilosVideo172\_Dotnet\_Grupo1 60](#_Toc124900927)

[Figura 63. Ejecución en consola de HilosVideo172\_Dotnet\_Grupo1 61](#_Toc124900928)

**ÍNDICE DE TABLAS**

[Tabla 1. Clases de System.Threading 7](#_Toc124900929)

[Tabla 2. Estructuras de la clase System.Threading 10](#_Toc124900930)

[Tabla 3. Enumeraciones clase System.Threading 11](#_Toc124900931)

[Tabla 4. Delegados en System.Threading 11](#_Toc124900932)

[Tabla 5. Propiedades de ThreadPool 13](#_Toc124900933)

[Tabla 6. Métodos de ThreadPool 13](#_Toc124900934)

[Tabla 7. Campos de ThreadState 16](#_Toc124900935)

[Tabla 8. Estados y acciones de ThreadState 17](#_Toc124900936)

[Tabla 9. Campos y acciones de ThreadPriority 18](#_Toc124900937)

[Tabla 10.Constructores de ThreadLocal<T> 18](#_Toc124900938)

[Tabla 11. Propiedades de ThreadLocal<T> 19](#_Toc124900939)

[Tabla 12.Métodos de la clase ThreadLocal<T> 19](#_Toc124900940)

[Tabla 13. Propiedades de un hilo en la clase Thread 23](#_Toc124900941)

[Tabla 14. Métodos de un hilo en la clase Thread 23](#_Toc124900942)

[Tabla 15. Excepciones en Hilos 33](#_Toc124900943)

[Tabla 16. Constructores de la clase Thread 35](#_Toc124900944)

[Tabla 17. Codificación del archivo Hilo.cs 43](file:///C:\Users\kenne\OneDrive\Escritorio\MONSTER\SEGUNDO_P\HILOS_DOTNET_2DO\DOCUMENTACION\HILOS_DOTNET_ANDRADE_MORENO_VACAS\SEGUNDO%20PROYECTO\DOCUMENTACION\HilosProyecto2_Dotnet_Grupo1.docx#_Toc124900945)

[Tabla 18. Codificación del archivo HiloControlador.cs 45](file:///C:\Users\kenne\OneDrive\Escritorio\MONSTER\SEGUNDO_P\HILOS_DOTNET_2DO\DOCUMENTACION\HILOS_DOTNET_ANDRADE_MORENO_VACAS\SEGUNDO%20PROYECTO\DOCUMENTACION\HilosProyecto2_Dotnet_Grupo1.docx#_Toc124900946)

[Tabla 19. Codificación del archivo Principal.cs 48](file:///C:\Users\kenne\OneDrive\Escritorio\MONSTER\SEGUNDO_P\HILOS_DOTNET_2DO\DOCUMENTACION\HILOS_DOTNET_ANDRADE_MORENO_VACAS\SEGUNDO%20PROYECTO\DOCUMENTACION\HilosProyecto2_Dotnet_Grupo1.docx#_Toc124900947)

[Tabla 20. Codificación del archivo Hilo.cs 54](file:///C:\Users\kenne\OneDrive\Escritorio\MONSTER\SEGUNDO_P\HILOS_DOTNET_2DO\DOCUMENTACION\HILOS_DOTNET_ANDRADE_MORENO_VACAS\SEGUNDO%20PROYECTO\DOCUMENTACION\HilosProyecto2_Dotnet_Grupo1.docx#_Toc124900948)

[Tabla 21. Codificación del archivo HiloControlador.cs 56](file:///C:\Users\kenne\OneDrive\Escritorio\MONSTER\SEGUNDO_P\HILOS_DOTNET_2DO\DOCUMENTACION\HILOS_DOTNET_ANDRADE_MORENO_VACAS\SEGUNDO%20PROYECTO\DOCUMENTACION\HilosProyecto2_Dotnet_Grupo1.docx#_Toc124900949)

[Tabla 22. Codificación del archivo Principal.cs 58](file:///C:\Users\kenne\OneDrive\Escritorio\MONSTER\SEGUNDO_P\HILOS_DOTNET_2DO\DOCUMENTACION\HILOS_DOTNET_ANDRADE_MORENO_VACAS\SEGUNDO%20PROYECTO\DOCUMENTACION\HilosProyecto2_Dotnet_Grupo1.docx#_Toc124900950)

# MARCO TEÓRICO

## INTRODUCCIÓN

Existen dos tipos de programas: los monotarea y los multitareas. Los programas monotarea solo pueden ejecutar una tarea a la vez y permiten a un usuario realizar una sola tarea a la vez. En el caso de un sistema multiusuario y monotarea, varios usuarios pueden ingresar, pero cada uno está limitado a realizar una tarea a la vez.

Por otro lado, los programas multitareas permiten que el usuario realice varias tareas simultáneamente. Por ejemplo, el usuario puede editar el código fuente de un programa mientras compila otro programa y al mismo tiempo recibir correos electrónicos en segundo plano. Estos programas suelen contar con interfaces gráficas que facilitan el intercambio rápido entre las tareas, lo que mejora la productividad del usuario.

## OBJETIVOS

### **OBJETIVO GENERAL**

* El objetivo es comprender y analizar el funcionamiento de monotarea y multitarea en DotNet (.Net) utilizando el patrón de diseño MVC, con el fin de aprender su implementación a partir del IDE de desarrollo Visual Studio 2022.

### **OBJETIVOS ESPECÍFICOS**

* Conocer y codificar ejercicios Thread en el entorno .Net para comprender las sincronizaciones de hilos utilizando el patrón de diseño MVC.
* Aprender los diferentes estados de los hilos en el entorno de DotNet mediante la aplicación del patrón de diseño MVC.
* Comprender la importancia de las condicionales al trabajar con hilos, según las necesidades específicas.

## HILOS

Un Thread (Hilo) es algo así como una CPU virtualizada que permite a una aplicación realizar varias tareas a la vez en paralelo, es parecido al concepto multitarea a nivel del sistema operativo pero los Threads se enfocan en subprocesos que pertenecen a un mismo proceso y la diferencia es que los Threads comparten espacio de memoria y los procesos no. [21]  
Para el caso del Sistema Operativo Windows, este administra todos los subprocesos para garantizar que puedan ejecutarse y realizar su trabajo, el SO se encarga de esa administración, de darle tiempo de ejecución en la CPU y cuando este tiempo de ejecución termina, el subproceso se detiene y Windows cambia a otro Thread, esto es a lo que se le conoce como cambio de contexto.  
Para poder utilizar Threads en nuestras apliaciones .NET podemos recurrir al espacio de nombres System.Threading en donde encontraremos la clase Thread con la que podremos crear nuevos Threads, gestionar su prioridad y obtener su estado. [1]

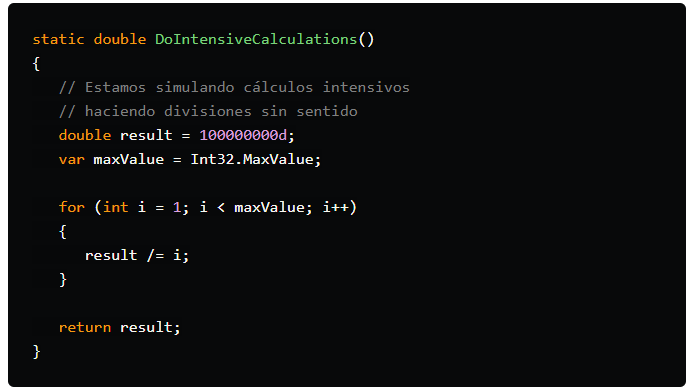


Figura 1. Ejemplo de la sintaxis de hilos

### **1.3.1. CLASE SYSTEM.THREADING**

Proporciona clases e interfaces que habilitan la programación multiproceso. Además de las clases para la sincronización de actividades de subprocesos y el acceso a datos ([Mutex](https://learn.microsoft.com/es-es/dotnet/api/system.threading.mutex?view=net-7.0), [Monitor](https://learn.microsoft.com/es-es/dotnet/api/system.threading.monitor?view=net-7.0), [Interlocked](https://learn.microsoft.com/es-es/dotnet/api/system.threading.interlocked?view=net-7.0), [AutoResetEvent](https://learn.microsoft.com/es-es/dotnet/api/system.threading.autoresetevent?view=net-7.0), etc.), este espacio de nombres incluye una clase [ThreadPool](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpool?view=net-7.0) que permite usar un grupo de subprocesos suministrados por el sistema y una clase [Timer](https://learn.microsoft.com/es-es/dotnet/api/system.threading.timer?view=net-7.0) que ejecuta métodos de devolución de llamada en subprocesos del grupo de subprocesos [2].

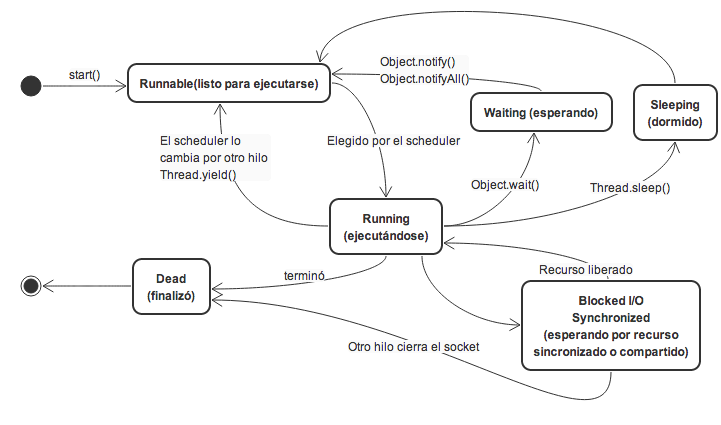


Figura 2. Uso de Paralelismo en clase System.Threading

Las siguientes clases pueden ser invocadas a partir de la declaración System.Threading:

Tabla 1. Clases de System.Threading

|  |  |
| --- | --- |
| CLASES | REPRESENTACIÓNES |
| ABANDONEDMUTEXEXCEPTION | Excepción que se produce cuando un subproceso adquiere un objeto [Mutex](https://learn.microsoft.com/es-es/dotnet/api/system.threading.mutex?view=net-7.0) que otro subproceso ha abandonado al salir sin liberarlo [2]. |
| ASYNCLOCAL<T> | Representa datos ambiente locales de un flujo de control asincrónico determinado, por ejemplo, un método asincrónico [2]. |
| AUTORESETEVENT | Representa un evento de sincronización de subprocesos que cuando se señala, se restablece automáticamente después de liberar un único subproceso en espera. Esta clase no puede heredarse [2]. |
| BARRIER | Habilita Habilita varias tareas para que colaboren en un algoritmo en paralelo en varias fases [2]. |
| BARRIERPOSTPHASEEXCEPTION | La excepción que se inicia cuando se produce un error en la acción posterior a la fase de [Barrier](https://learn.microsoft.com/es-es/dotnet/api/system.threading.barrier?view=net-7.0) [2]. |
| CANCELLATIONTOKENSOURCE | Señala un objeto [CancellationToken](https://learn.microsoft.com/es-es/dotnet/api/system.threading.cancellationtoken?view=net-7.0) que debe cancelarse [2]. |
| COMPRESSEDSTACK | Proporciona métodos para la configuración y captura de la pila comprimida en el subproceso actual. Esta clase no puede heredarse [2]. |
| COUNTDOWNEVENT | Representa una primitiva de sincronización que está señalada cuando su recuento alcanza el valor cero [2]. |
| EVENTWAITHANDLE | Representa un evento de sincronización de subprocesos [2]. |
| EXECUTIONCONTEXT | Administra el contexto de ejecución para el subproceso actual. Esta clase no puede heredarse [2]. |
| HOSTEXECUTIONCONTEXT | Encapsula y difunde el contexto de ejecución del host en todos los subprocesos [2]. |
| HOSTEXECUTIONCONTEXTMANAGER | Proporciona la funcionalidad que permite que un host de Common Language Runtime participe en el flujo, o migración, del contexto de ejecución [2]. |
| INTERLOCKED | Proporciona operaciones atómicas para las variables compartidas por varios subprocesos [2]. |
| LAZYINITIALIZER | Proporciona rutinas de inicialización diferida. |
| LOCKRECURSIONEXCEPTION | Excepción que se inicia cuando la entrada recursiva en un bloqueo no es compatible con la directiva de recursión del bloqueo [2]. |
| MANUALRESETEVENT | Representa un evento de sincronización de subprocesos que, cuando se señale, se debe restablecer manualmente. Esta clase no puede heredarse [2]. |
| MANUALRESETEVENTSLIM | Representa un evento de sincronización de subprocesos que, cuando se señale, se debe restablecer manualmente. Esta clase es una alternativa ligera a [ManualResetEvent](https://learn.microsoft.com/es-es/dotnet/api/system.threading.manualresetevent?view=net-7.0) [2]. |
| MONITOR | Proporciona un mecanismo que sincroniza el acceso a los objetos [2]. |
| MUTEX | Primitiva de sincronización que puede usarse también para la sincronización entre procesos [2]. |
| OVERLAPPED | Proporciona una representación administrada de una estructura OVERLAPPED de Win32, incluidos los métodos para transferir información de una instancia [Overlapped](https://learn.microsoft.com/es-es/dotnet/api/system.threading.overlapped?view=net-7.0) a una estructura [NativeOverlapped](https://learn.microsoft.com/es-es/dotnet/api/system.threading.nativeoverlapped?view=net-7.0) [2]. |
| PERIODICTIMER | Proporciona un temporizador periódico que permite esperar de forma asincrónica para los tics del temporizador [2]. |
| PREALLOCATEDOVERLAPPED | Representa el estado preasignado para las operaciones de E/S superpuestas nativas [2]. |
| READERWRITERLOCK | Define un bloqueo que admite un escritor y varios lectores [2]. |
| READERWRITERLOCKSLIM | Representa un bloqueo que se utiliza para administrar el acceso a un recurso y que permite varios subprocesos para la lectura o acceso exclusivo para la escritura [2]. |
| REGISTEREDWAITHANDLE | Representa un identificador que se registró al llamar a [RegisterWaitForSingleObject(WaitHandle, WaitOrTimerCallback, Object, UInt32, Boolean)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpool.registerwaitforsingleobject?view=net-7.0" \l "system-threading-threadpool-registerwaitforsingleobject(system-threading-waithandle-system-threading-waitortimercallback-system-object-system-uint32-system-boolean)). Esta clase no puede heredarse [2]. |
| SEMAPHORE | Limita el número de subprocesos que pueden tener acceso a un recurso o grupo de recursos simultáneamente [2]. |
| SEMAPHOREFULLEXCEPTION | Excepción que se produce cuando se llama al método [Release](https://learn.microsoft.com/es-es/dotnet/api/system.threading.semaphore.release?view=net-7.0) en un semáforo cuyo recuento ya se encuentra en el máximo [2]. |
| SEMAPHORESLIM | Representa una alternativa ligera a [Semaphore](https://learn.microsoft.com/es-es/dotnet/api/system.threading.semaphore?view=net-7.0) que limita el número de subprocesos que pueden tener acceso a un recurso o grupo de recursos simultáneamente [2]. |
| SYNCHRONIZATIONCONTEXT | Proporciona la funcionalidad básica para difundir un contexto de sincronización en varios modelos de sincronización [2]. |
| SYNCHRONIZATIONLOCKEXCEPTION | Excepción que se produce cuando un método requiere que el llamador sea propietario del bloqueo en un Monitor dado y un llamador al que no pertenece ese bloqueo llama al método [2]. |
| THREAD | Crea y controla un subproceso, establece su prioridad y obtiene su estado [2]. |
| THREADLOCATE<P> | Proporciona almacenamiento local de los datos de un subproceso [2]. |
| THREADPOOL | Proporciona un grupo de subprocesos que pueden usarse para ejecutar tareas, exponer elementos de trabajo, procesar la E/S asincrónica, esperar en nombre de otros subprocesos y procesar temporizadores [2]. |
| THREADPOOLBOUNDHANDLE | Representa un identificador de E/S que se enlaza al grupo de subprocesos de sistema y permite que los componentes de bajo nivel reciban notificaciones para operaciones de E/S asincrónicas [2]. |
| TIMER | Proporciona un mecanismo para ejecutar un método en un subproceso del grupo de subprocesos en intervalos especificados. Esta clase no puede heredarse [2]. |
| VOLATILE | Contiene los métodos para realizar operaciones de memoria volátil [2]. |
| WAITHANDLE | Encapsula los objetos específicos del sistema operativo que esperan obtener un acceso exclusivo a los recursos compartidos [2]. |
| WAITHANDLECANNOTBEOPENED | La excepción que se produce cuando se intenta abrir una exclusión mutua del sistema, un semáforo o un identificador de espera de eventos que no existe [2]. |
| WAITHANDLEEXTENSIONS | Proporciona métodos útiles para trabajar con un controlador seguro para un identificador de espera [2]. |

Las estructuras que son utilizadas en la clase System.Threading de Dotnet son:

Tabla 2. Estructuras de la clase System.Threading

|  |  |
| --- | --- |
| ESTRUCTURAS | APLICACIÓN |
| ASYNCFLOWCONTROL | Proporciona la funcionalidad para restaurar la migración, o el flujo, del contexto de ejecución entre los subprocesos [2]. |
| ASUNCLOCALVALUE<T> | Clase que proporciona información de cambio de datos a las instancias [AsyncLocal<T>](https://learn.microsoft.com/es-es/dotnet/api/system.threading.asynclocal-1?view=net-7.0) que se registran para las notificaciones de cambios [2]. |
| CANCELLATIONTOKEN | Propaga la notificación de que las operaciones deberían cancelarse [2]. |
| CANCELATIONTOKENREGISTRATION | Representa un delegado de devolución de llamada que se ha registrado con un token [CancellationToken](https://learn.microsoft.com/es-es/dotnet/api/system.threading.cancellationtoken?view=net-7.0) [2]. |
| LOOKCOOKIE | Define el bloqueo que implementa la semántica de un sistema de escritura y varios sistemas de lectura. Se trata de un tipo de valor [2]. |
| NATIVEOVERLAPPED | Proporciona un diseño explícito visible desde el código no administrado y que tendrá el mismo diseño que la estructura OVERLAPPED de Win32 con campos reservados adicionales al final [2]. |
| SPINLOCK | Proporciona una primitiva de bloqueo de exclusión mutua donde un subproceso que intenta adquirir el bloqueo espera en un bucle y realiza comprobaciones repetidamente hasta que haya un bloqueo disponible [2]. |
| SPINWWAIT | Proporciona compatibilidad para la espera basada en ciclos [2]. |

System.Threading tiene una Interfaz (ITHREADPOOLWORKITEM) la cual representa un elemento de trabajo que THREADPOOL puede ejecutar [2].

Además de Interfaces System.Threading cuenta con enumeraciones que muestra los estados de los hilos estableciéndose a partir de una especificación de Instancia [2].

Tabla 3. Enumeraciones clase System.Threading

|  |  |
| --- | --- |
| ENUMERACIONES | DEFINICIÓNES |
| APARTMENTSTATE | Especifica el estado de apartamento de un [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) [2]. |
| EVENTRESETMODE | Indica si un objeto [EventWaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.eventwaithandle?view=net-7.0) se restablece automática o manualmente después de recibir una señal [2]. |
| LAZYTHREADSAFETYMODE | Especifica cómo una instancia [Lazy<T>](https://learn.microsoft.com/es-es/dotnet/api/system.lazy-1?view=net-7.0) sincroniza el acceso entre varios subprocesos [2]. |
| LOCKREVURSIONPOLICY | Especifica si el mismo subproceso puede entrar varias veces en un bloqueo [2]. |
| THREADPRIORITY | Especifica la prioridad de programación de [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) [2]. |
| THREADSTATE | Especifica los estados de ejecución de [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) [2]. |

Se incorporan en SystemThreading los Delegados los cuales almacenen la referencia de un método con una estructura definida (en cuanto al tipo de dato que devuelve y cantidad de parámetros) [3].

Tabla 4. Delegados en System.Threading

|  |  |
| --- | --- |
| DELEGADOS | DEFINICIÓNES |
| CONTEXTCALLBACK | Representa un método al que llamar dentro de un nuevo contexto [2]. |
| IOCOMPLETIONCALLBACK | Recibe el código de error, el número de bytes y el tipo de valor superpuesto cuando finaliza una operación de E/S en el grupo de subprocesos [2]. |
| PARAMETERIZEDTHREADSTART | Representa el método que se ejecuta en [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) [2]. |
| SENDORPOSTCALLBACK | Representa el método al que hay que llamar cuando se va a enviar un mensaje a un contexto de sincronización [2]. |
| THREADEXCEPTIONEVENTHANDLER | Representa el método que controlará el evento [ThreadException](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.application.threadexception?view=net-7.0) de [Application](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.application?view=net-7.0) [2]. |
| THREADSTART | Representa el método que se ejecuta en [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) [2]. |
| TIMERCALLBACK | Representa el método que controla las llamadas de un [Timer](https://learn.microsoft.com/es-es/dotnet/api/system.threading.timer?view=net-7.0) [2]. |
| WAITCALLBACK | Representa un método de devolución de llamada que debe ejecutar un subproceso del grupo de subprocesos [2]. |
| WAITORTIMERCALLBACK | Representa un método al que se debe llamar cuando un [WaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.waithandle?view=net-7.0) está señalado o se agota su tiempo de espera [2]. |

### **1.3.2. THREADSTART**

Proveniente de la clase System.Threading, representa el método que ejecuta el Hilo a partir de public delegate void ThreadStart() [4].

Cuando se crea un subproceso administrado, el método que se ejecuta en el subproceso se representa mediante un [ThreadStart](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstart?view=net-7.0) delegado o un [ParameterizedThreadStart](https://learn.microsoft.com/es-es/dotnet/api/system.threading.parameterizedthreadstart?view=net-7.0) delegado que se pasa al [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.-ctor?view=net-7.0) constructor. El subproceso no comienza a ejecutarse hasta que se llama al [Thread.Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0) método . La ejecución comienza en la primera línea del método representado por el [ThreadStart](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstart?view=net-7.0) delegado o [ParameterizedThreadStart](https://learn.microsoft.com/es-es/dotnet/api/system.threading.parameterizedthreadstart?view=net-7.0) [4].

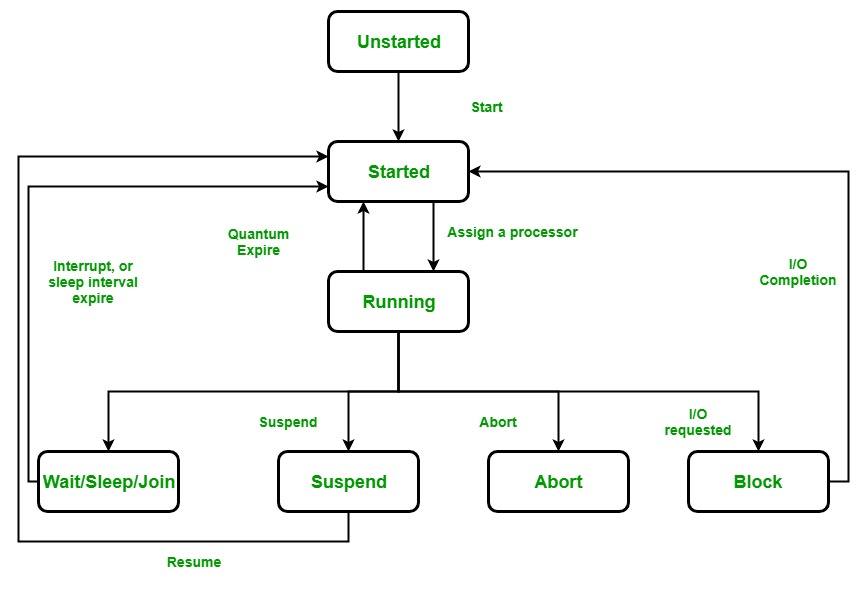


Figura 3. Recorrido de ThreadStart

### **1.3.3. THREADPOOL**

Proveniente de la clase System.Threading proporciona un grupo de subprocesos que pueden usarse para ejecutar tareas, exponer elementos de trabajo, procesar la E/S asincrónica, esperar en nombre de otros subprocesos y procesar temporizadores [5].

Cuando el grupo de subprocesos reutiliza un subproceso, no borra los datos en el almacenamiento local del subproceso ni en campos marcados con el [ThreadStaticAttribute](https://learn.microsoft.com/es-es/dotnet/api/system.threadstaticattribute?view=net-7.0) atributo . Por lo tanto, cuando un método examina el almacenamiento local del subproceso o los campos marcados con el [ThreadStaticAttribute](https://learn.microsoft.com/es-es/dotnet/api/system.threadstaticattribute?view=net-7.0) atributo , los valores que encuentra podrían dejarse de un uso anterior del subproceso del grupo de subprocesos [5].



Figura 4. Tareas de ThreadPool

Las propiedades de ThreadPool se encuentran detalladas a continuación:

Tabla 5. Propiedades de ThreadPool

|  |  |
| --- | --- |
| PROPIEDADES | DEFINICIÓNES |
| COMPLETEDWORKITEMCOUNT | Obtiene el número de elementos de trabajo que se han procesado hasta ahora [5]. |
| PENDINGWORKITEMCOUNT | Obtiene el número de elementos de trabajo que se encuentran actualmente en cola para procesarse [5]. |
| THREADCOUNT | Obtiene el número de subprocesos del grupo de subprocesos que existen actualmente [5]. |

Los métodos que utiliza ThreadPool vienen detallados a continuación.

Tabla 6. Métodos de ThreadPool

|  |  |
| --- | --- |
| MÉTODOS | DEFINICIÓNES |
| BINDHANDLE(INTPTR) | Obsoleto, Enlaza un identificador del sistema operativo con ThreadPool [5]. |
| BINDHANDLE(SAFEHANDLE) | Enlaza un identificador del sistema operativo a [ThreadPool](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpool?view=net-7.0) [5]. |
| GETAVAILABLETHREADS(INT32,INT32) | Recupera la diferencia entre el número máximo de subprocesos de grupo de subprocesos devuelto por el método [GetMaxThreads(Int32, Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpool.getmaxthreads?view=net-7.0" \l "system-threading-threadpool-getmaxthreads(system-int32@-system-int32@)) y el número activo actualmente [5]. |
| GETMAXTHREADS(INT32,INT32) | Recupera el número de solicitudes al grupo de subprocesos que pueden estar activas al mismo tiempo. Todas las solicitudes que pasen de ese número permanecen en la cola hasta que haya disponibles subprocesos de grupo de subprocesos [5]. |
| GETMINTHREADS(INT32,INT32) | Recupera el número mínimo de subprocesos que el grupo de subprocesos crea a petición, según se realizan nuevas solicitudes, antes de conmutar a un algoritmo para administrar la creación y destrucción de subprocesos [5]. |
| QUEUEUSERWORKITEM(WAIRCALLBACK) | Pone en cola un método para su ejecución. El método se ejecuta cuando hay disponible un subproceso de grupo de subprocesos [5]. |
| QUEUEUSERWORKITEM(WAITCALLBACK,OBJECT) | Pone un método en cola para su ejecución y especifica un objeto que contiene los datos que debe usar el método. El método se ejecuta cuando hay disponible un subproceso de grupo de subprocesos [5]. |
| REGISTERWAITFORSINGLEOBJECT(WAITHANDLE) | Registra un delegado para que espere a la clase [WaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.waithandle?view=net-7.0) y especifica un entero de 64 bits con signo como tiempo de espera, en milisegundos [5]. |
| SETMAXTHREADS(INT32,INT32) | Establece el número de solicitudes al grupo de subprocesos que pueden estar activas al mismo tiempo. Todas las solicitudes que pasen de ese número permanecen en la cola hasta que haya disponibles subprocesos de grupo de subprocesos [5]. |
| SETMINTHREADS(INT32,INT32) | stablece el número mínimo de subprocesos que el grupo de subprocesos crea a petición, según se realizan nuevas solicitudes, antes de conmutar a un algoritmo para administrar la creación y destrucción de subprocesos [5]. |
| UNSAFEQUEUENATIVEOVERLAPPED(NATIVE) | Pone en cola el objeto del elemento de trabajo especificado en el grupo de subprocesos [5]. |
| UNSAFEQUEUENATIVEOVERLAPPED(THREADPOOL) | Pone en cola el delegado especificado en el grupo de subprocesos, pero no propaga la pila de llamadas al subproceso de trabajo [5]. |
| UNSAFEQUEUENATIVEOVERLAPPED(NATIVE) | Registra un delegado para que espere a la clase [WaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.waithandle?view=net-7.0) y usa un entero de 32 bits con signo como tiempo de espera, en milisegundos. Este método no propaga la pila de llamadas al subproceso de trabajo [5]. |
| UNSAFEQUEUEUSERWAITFORSINGLEOBJECT(  WAITHANDLE,WAITPRTIMERCALLBACK,  OBJECT,INT32,BOOLEAN) | Registra un delegado para que espere a la clase [WaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.waithandle?view=net-7.0) y especifica un entero de 64 bits con signo como tiempo de espera, en milisegundos. Este método no propaga la pila de llamadas al subproceso de trabajo [5]. |
| UNSAFEREGISTERWAITFORSINGLEOBJECT(  WAITHANDLE,WAITPRTIMERCALLBACK,  OBJECT,INT32,BOOLEAN) | Registra un delegado para que espere a la clase [WaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.waithandle?view=net-7.0) y especifica un valor [TimeSpan](https://learn.microsoft.com/es-es/dotnet/api/system.timespan?view=net-7.0) como tiempo de espera. Este método no propaga la pila de llamadas al subproceso de trabajo [5]. |
| UNSAFEREGISTERWAITFORSINGLEOBJECT(  WAITHANDLE,WAITPRTIMERCALLBACK,  UINT32,BOOLEAN) | Registra un delegado para que espere a la clase [WaitHandle](https://learn.microsoft.com/es-es/dotnet/api/system.threading.waithandle?view=net-7.0) y especifica un entero de 32 bits sin signo como tiempo de espera, en milisegundos. Este método no propaga la pila de llamadas al subproceso de trabajo [5]. |

Por su parte el grupo de subprocesos proporciona nuevos subprocesos de trabajo o subprocesos de finalización de E/S a petición hasta que alcanza el máximo para cada categoría. Cuando se alcanza un máximo, el grupo de subprocesos puede crear subprocesos adicionales en esa categoría o esperar hasta que se completen algunas tareas. A partir de .NET Framework 4, el grupo de subprocesos crea y destruye subprocesos de trabajo para optimizar el rendimiento, definido como el número de tareas que se completan por unidad de tiempo. Si hay demasiados pocos subprocesos, puede que los recursos disponibles no se usen de manera óptima, mientras que, si hay demasiados subprocesos, puede aumentar la contención de recursos [5].

### **1.3.4. THREADSTATE**

Proveniente de System.Threading, contiene una combinación de Hilos bit\*bie a partir de la llamada a su clase public enum ThreadState [6].

La ThreadState enumeración define un conjunto de todos los estados de ejecución posibles para los subprocesos. Solo es de interés en algunos escenarios de depuración. El código nunca debe usar el estado del subproceso para sincronizar las actividades de los subprocesos [6].

Una vez creado un subproceso, se encuentra al menos en uno de los estados hasta que finaliza. Los subprocesos creados en Common Language Runtime están inicialmente en estado [Unstarted](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-unstarted) , mientras que los subprocesos externos o no administrados que entran en tiempo de ejecución ya están en estado [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running) . Un subproceso pasa del [Unstarted](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-unstarted) estado al [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running) estado mediante una llamada a [Thread.Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0). Una vez que un subproceso abandona el estado [Unstarted](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-unstarted) como resultado de una llamada a [Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0), nunca puede volver al estado [Unstarted](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-unstarted) [6].

Tabla 7. Campos de ThreadState

|  |  |
| --- | --- |
| CAMPOS | DEFINICIÓNES |
| ABORTED | El estado del subproceso incluye [AbortRequested](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-abortrequested) y el subproceso está ahora inactivo, pero su estado no ha cambiado todavía a [Stopped](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-stopped) [6]. |
| ABORTREQUESTED | Se ha invocado al 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)) en el subproceso, pero el subproceso aún no ha recibido la excepción [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception?view=net-7.0) pendiente que intentará finalizarlo [6]. |
| BACKGROUND | El subproceso está ejecutándose como subproceso en segundo plano, por oposición a un subproceso en primer plano. Para controlar este estado, hay que establecer la propiedad [IsBackground](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isbackground?view=net-7.0" \l "system-threading-thread-isbackground) [6]. |
| RUNNING | El subproceso se ha iniciado y aún no se ha detenido [6]. |
| STOPPED | El subproceso se ha detenido [6]. |
| STOPREQUESTED | Se ha solicitado que el subproceso se detenga. Esto último sólo se refiere al uso interno [6]. |
| SUSPENDED | El subproceso se ha suspendido [6]. |
| SUSPENDREQUESTED | Se ha solicitado que el subproceso se suspenda [6]. |
| UNSTARTED | No se ha invocado al método [Start()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0" \l "system-threading-thread-start) en el subproceso [6]. |
| WAITSLEEPJOIN | Subproceso bloqueado. Este podría ser el resultado de llamar a [Sleep(Int32)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep?view=net-7.0" \l "system-threading-thread-sleep(system-int32)) o [Join()](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join?view=net-7.0" \l "system-threading-thread-join), de solicitar un bloqueo (por ejemplo, llamando a [Enter(Object)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.monitor.enter?view=net-7.0" \l "system-threading-monitor-enter(system-object)) o [Wait(Object, Int32, Boolean)](https://learn.microsoft.com/es-es/dotnet/api/system.threading.monitor.wait?view=net-7.0" \l "system-threading-monitor-wait(system-object-system-int32-system-boolean))) o de esperar en un objeto de sincronización de subprocesos como [ManualResetEvent](https://learn.microsoft.com/es-es/dotnet/api/system.threading.manualresetevent?view=net-7.0) [6]. |

Un subproceso puede estar en más de un estado en un momento dado. Por ejemplo, si un subproceso está bloqueado en una llamada a [Monitor.Wait](https://learn.microsoft.com/es-es/dotnet/api/system.threading.monitor.wait?view=net-7.0)y otro subproceso llama [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0) al subproceso bloqueado, el subproceso bloqueado estará en los [WaitSleepJoin](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-waitsleepjoin) estados y [AbortRequested](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-abortrequested) al mismo tiempo. En este caso, tan pronto como el subproceso vuelva de la llamada a [Monitor.Wait](https://learn.microsoft.com/es-es/dotnet/api/system.threading.monitor.wait?view=net-7.0) o se interrumpa, recibirá el objeto [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception?view=net-7.0) para comenzar a anularse. No todas las combinaciones de ThreadState valores son válidas; por ejemplo, un subproceso no puede estar en los [Aborted](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-aborted) estados y [Unstarted](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0" \l "system-threading-threadstate-unstarted) [6] .

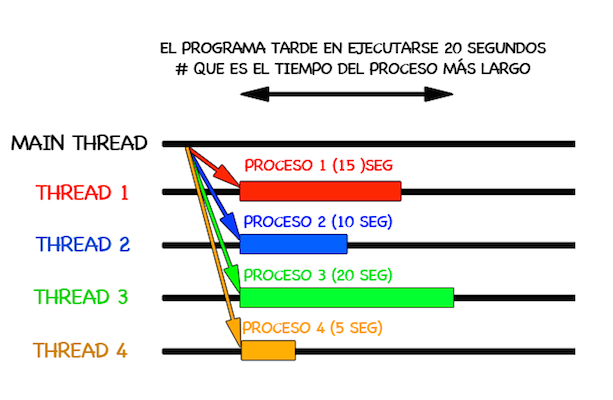


Figura 5. Ejecución a partir de MAIN THREAD

En la tabla siguiente se muestran las acciones que provocan un cambio de estado [6].

Tabla 8. Estados y acciones de ThreadState

|  |  |
| --- | --- |
| ESTADO DEL HILO | ACCIÓN |
| UNSTARTED | Se crea un subproceso en Common Language Runtime [6]. |
| RUNNING | El [Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0) método no devuelve hasta que se haya iniciado la ejecución del nuevo subproceso. No hay ninguna manera de saber en qué momento se iniciará la ejecución del nuevo subproceso durante la llamada a [Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start?view=net-7.0) [6]. |
| WAITSLEEPJOIN | El subproceso llama a [Sleep](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep?view=net-7.0) [6]. |
| WAITSLEEPJOIN | El subproceso llama a [Monitor.Wait](https://learn.microsoft.com/es-es/dotnet/api/system.threading.monitor.wait?view=net-7.0) en otro objeto [6]. |
| WAITSLEEPJOIN | El subproceso llama a [Join](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join?view=net-7.0) en otro subproceso [6]. |
| RUNNING | Otra llamada de subproceso [Interrupt](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt?view=net-7.0) [6]. |
| SUSPENDREQUESTED | Otra llamada de subproceso [Suspend](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.suspend?view=net-7.0) [6]. |
| SUSPENDED | El subproceso responde a una solicitud [Suspend](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.suspend?view=net-7.0) [6]. |
| RUNNING | Otra llamada de subproceso [Resume](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.resume?view=net-7.0) [6]. |
| ABORTREQUESTED | Otra llamada de subproceso  [6]. |
| STOPPED | El subproceso responde a una [Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort?view=net-7.0) solicitud [6]. |
| STOPPED | Se termina un subproceso [6]. |

### **1.3.5. THREADPRIORITY**

[ThreadPriority](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpriority?view=net-7.0) define el conjunto de todos los valores posibles para una prioridad de subproceso. Las prioridades de subproceso especifican la prioridad relativa de un subproceso frente a otro [7].

Cada subproceso tiene una prioridad asignada. Los subprocesos creados en el tiempo de ejecución se asignan inicialmente a la Normal prioridad, mientras que los subprocesos creados fuera del tiempo de ejecución conservan su prioridad anterior cuando entran en tiempo de ejecución. Puede obtener y establecer la prioridad de un subproceso accediendo a su [Priority](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.priority?view=net-7.0) propiedad [7].

Tabla 9. Campos y acciones de ThreadPriority

|  |  |
| --- | --- |
| CAMPOS | ACCIÓNES |
| ABOVENORMAL | [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) puede programarse después de los subprocesos con prioridad Highest y antes que los subprocesos con prioridad Normal [7]. |
| BELOWNORMAL | [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) puede programarse después de los subprocesos con prioridad Normal y antes que los subprocesos con prioridad Lowest [7]. |
| HIGHEST | [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) puede programarse antes que los subprocesos que tengan cualquier otra prioridad [7]. |
| LOWEST | [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) puede programarse después de los subprocesos que tengan cualquier otra prioridad [7]. |
| NORMAL | [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread?view=net-7.0) puede programarse después de los subprocesos con prioridad AboveNormal y antes que los subprocesos con prioridad BelowNormal. Los subprocesos tienen prioridad Normal de forma predeterminada [7]. |

La prioridad de un subproceso no afecta al estado del subproceso; el estado del subproceso debe ser [Running](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0#system-threading-threadstate-running) antes de que el sistema operativo pueda programarlo [7].

### **1.3.6. THREADLOCAL<T>**

Procedente de System.Threading, proporciona almacenamiento local de los datos dentro de un subproceso [8].

Los constructores que contiene esta clase son los siguientes:

Tabla 10.Constructores de ThreadLocal<T>

|  |  |
| --- | --- |
| CONSTRUCTORES | INSTANCIAS |
| THREADLOCAL<T>() | Inicializa la instancia ThreadLocal<T> [8] |
| THREADLOCAL<T>(BOOLEAN) | Inicializa la instancia [ThreadLocal<T>](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1?view=net-7.0) y especifica si todos los valores son accesibles desde cualquier subproceso [8]. |
| THREADLOCAL<T>(FUNC<T>) | Inicializa una instancia de [ThreadLocal<T>](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1?view=net-7.0) con la función especificada por el parámetro valueFactory [8]. |
| THREADLOCAL<T>(FUNC<T>,BOOLEAN) | Inicializa la instancia [ThreadLocal<T>](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1?view=net-7.0) con la función valueFactory especificada y una marca que indica si todos los valores son accesibles desde cualquier subproceso [8]. |

Las propiedades que contiene esta clase son las siguientes:

Tabla 11. Propiedades de ThreadLocal<T>

|  |  |
| --- | --- |
| PROPIEDADES | DEFINICIÓN |
| ISVALUECREATED | Obtiene un valor que indica si [Value](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1.value?view=net-7.0" \l "system-threading-threadlocal-1-value) se inicializa en el subproceso actual [8]. |
| VALUE | Obtiene o establece el valor de esta instancia del subproceso actual [8]. |
| VALUES | Obtiene una lista que contiene los valores almacenados por todos los subprocesos a los que ha accedido esta instancia [8]. |

Los métodos que contiene esta clase son las siguientes:

Tabla 12.Métodos de la clase ThreadLocal<T>

|  |  |
| --- | --- |
| MÉTODOS | DEFINICIÓN |
| DISPOSE() | Obtiene un valor que indica si [Value](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1.value?view=net-7.0" \l "system-threading-threadlocal-1-value) se inicializa en el subproceso actual [8]. |
| DISPOSE(BOOLEAN) | Obtiene o establece el valor de esta instancia del subproceso actual [8]. |
| EQUALS(OBJECT) | Obtiene una lista que contiene los valores almacenados por todos los subprocesos a los que ha accedido esta instancia [8]. |
| FINALIZE() | Libera los recursos que usa la instancia [ThreadLocal<T>](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1?view=net-7.0) [8]. |
| GETTYPE() | Obtiene el Type de la instancia actual(Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) [8]. |
| MEMBERWISECLONE() | Crea una copia superficial del Object actual (Heredado de [Object](https://learn.microsoft.com/es-es/dotnet/api/system.object?view=net-7.0)) [8]. |
| TOSTRING() | Crea y devuelve una representación de cadena de esta instancia del subproceso actual [8]. |

### **1.3.7. VERSIONAMIENTO**

A continuación, se mostrará el versionamiento necesario para el respectivo correcto funcionamiento de propiedades y métodos que debe tener Dotnet [8].

Texto

Descripción generada automáticamente

Figura 6. Versionamiento Requerido para Threads de Dotnet

### **1.3.8. CREACIÓN DE SUBPROCESOS EN DOTNET**

Cuando se crea un proceso de sistema operativo, el sistema operativo inserta un subproceso para ejecutar el código en dicho proceso, incluido cualquier dominio de aplicación original. Desde ese momento, los dominios de aplicación se pueden crear y destruir sin que se cree o destruya necesariamente ningún subproceso del sistema operativo. Si el código ejecutado es código administrado, se puede obtener un objeto [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread) para el subproceso que se ejecuta en el dominio de aplicación actual mediante la recuperación de la propiedad [CurrentThread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.currentthread) estática de tipo [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread). En este tema se describe la creación de subprocesos y se analizan las alternativas para pasar datos al procedimiento de los subprocesos [9].

Al crear un objeto [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread), se crea un subproceso administrado. La clase [Thread](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread) tiene constructores que adoptan un delegado [ThreadStart](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstart) o un delegado [ParameterizedThreadStart](https://learn.microsoft.com/es-es/dotnet/api/system.threading.parameterizedthreadstart); el delegado ajusta el método invocado por el nuevo subproceso cuando llama al método [Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start). Al llamar a [Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start) más de una vez, se inicia [ThreadStateException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstateexception) [9].

El método [Start](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.start) se devuelve inmediatamente, a menudo antes de que se haya iniciado realmente el nuevo subproceso. Puede usar las propiedades [ThreadState](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.threadstate) y [IsAlive](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.isalive) para determinar el estado del subproceso en cualquier momento, pero estas propiedades nunca deben utilizarse para sincronizar las actividades de los subprocesos [9].

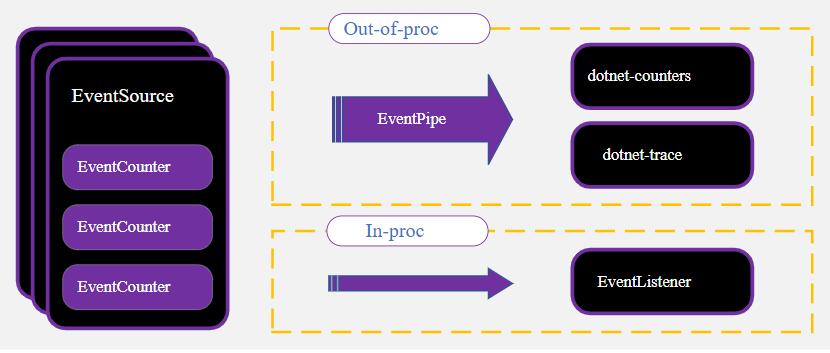


Figura 7. Creación Subprocesos Dotnet

### **1.3.9. DESTRUCCIÓN DE SUBPROCESOS EN DOTNET**

Para finalizar la ejecución del subproceso, normalmente se usa el [modelo de cancelación cooperativa](https://learn.microsoft.com/es-es/dotnet/standard/threading/cancellation-in-managed-threads). A veces no es posible detener un subproceso de forma cooperativa, ya que ejecuta código de terceros no diseñado para la cancelación cooperativa. El método [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) de .NET Framework se puede utilizar para terminar un subproceso administrado forzosamente. Cuando se llama a [Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort), Common Language Runtime inicia una clase [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception) en el subproceso de destino, que este último puede detectar. Para obtener más información, vea [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort). El método [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) no se admite en .NET 5 (incluido .NET Core) y versiones posteriores. Si tiene que terminar la ejecución de código de terceros de forma forzada en .NET 5 y versiones posteriores, ejecútelo en el proceso independiente y use [Process.Kill](https://learn.microsoft.com/es-es/dotnet/api/system.diagnostics.process.kill) [10].

Una vez que se anula un subproceso, no se puede reiniciar.

El método [Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) no hace que el subproceso se anule inmediatamente, porque el subproceso de destino puede detectar [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception) y ejecutar cantidades arbitrarias de código en un bloque finally. Puede llamar a [Thread.Join](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join) si tiene que esperar hasta que haya finalizado el subproceso. [Thread.Join](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join) es una llamada de bloqueo que no realiza ninguna devolución hasta que el subproceso se haya dejado de ejecutar realmente o hasta que haya transcurrido un intervalo de tiempo de espera opcional. El subproceso anulado podría llamar al método [ResetAbort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.resetabort) o llevar a cabo el procesamiento sin enlazar en un bloqueo finally, por lo que si no especifica un tiempo de espera, no se garantiza que la espera finalice [10].

Los subprocesos que esperan una llamada al método [Thread.Join](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.join) pueden interrumpirse con otros subprocesos que llaman a [Thread.Interrupt](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt) [10].

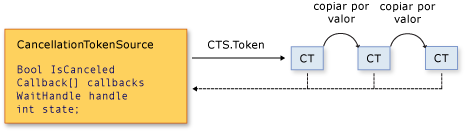


Figura 8. Llamada a callback desde TokenThread

Si espera que se anule el subproceso, como resultado de una llamada a [Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) desde su propio código o como resultado de la descarga de un dominio de aplicación en que se ejecuta el subproceso ([AppDomain.Unload](https://learn.microsoft.com/es-es/dotnet/api/system.appdomain.unload) usa [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) para terminar los subprocesos), el subproceso debe controlar [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception) y realizar cualquier procesamiento final en una cláusula finally [10].

### **1.3.10. VENTAJAS Y DESVENTAJAS DE LOS HILOS**

#### Ventajas

* Se tarda menos tiempo en crear un hilo de una tarea existente que en crear un nuevo proceso [11].
* Se tarda menos tiempo en terminar un hilo que en terminar un proceso [11].
* Se tarda menos tiempo en cambiar entre dos hilos de una misma tarea que en cambiar entre dos procesos (porque los recursos no cambian, por ejemplo) [11].
* Es más sencillo la comunicación (paso de mensajes, por ejemplo) entre hilos de una misma tarea que entre diferentes procesos [11].
* Cuando se cambia de un proceso a otro, tiene que intervenir el núcleo del sistema operativo para que haya protección. Cuando se cambia de un hilo a otro, puesto que la asignación de recursos es la misma, no hace falta que intervenga el sistema operativo [11].

#### Desventajas

* Una mala sincronización puede llevar a conducir a inconsistencia de datos [12].
* Esta limitado a capacidades del sistema en donde se corre [12].
* Mientras los procesos mantienen su propio espacio de direcciones y entorno de operaciones, los hilos dependen de un programa padre en lo que se refiere a recursos de ejecución [12].

### **1.3.11. CICLO DE VIDA DE UN HILO**

El uso de los hilos depende del lenguaje de programación o en ocasiones el IDE de desarrollo que se esté utilizando, para el caso de Dotnet. El ciclo de vida de un hilo comienza cuando se crea un objeto de la clase System.Threading.Thread y termina cuando el hilo se termina o finaliza la ejecución [13].

Sin embargo, se puede manipular el ciclo de vida de un hilo a partir de sus representaciones mediante cuatro diferentes estados:

1. **Estado no iniciado:** Es la situación en la que se crea la instancia de la clase Thread, pero el método de inicio no se llama [13].
2. **Estado listo:** Es la situación en la que el hilo está listo para funcionar y esperando el ciclo de la CPU [13].
3. **Estado no ejecutable:** Un hilo no es ejecutable cuando el método (Sleep) ha sido llamado, se ha llamado el método de espera bloqueado por operaciones de (E/S) (Entrada/Salida) [13].
4. **Estado muerto:** Es la situación en la que el hilo ha completado su ejecución o la misma ha sido abortada [13].

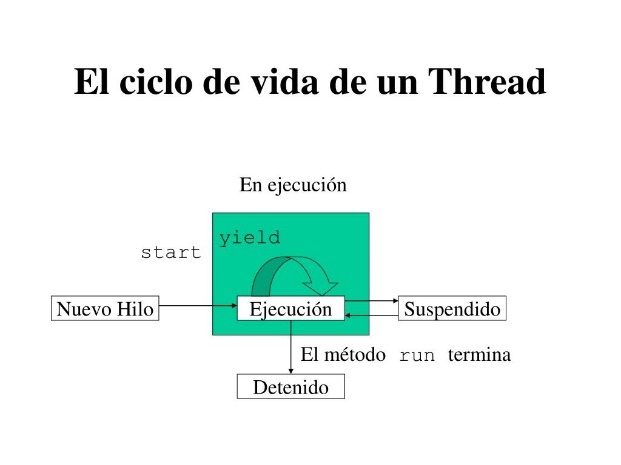


Figura 9. Ciclo de vida de un Hilo

### **1.3.12. PROPIEDADES Y MÉTODOS DE LA CLASE THREAD**

La clase Thread contiene propiedades y métodos para el acceso a los diferentes estados dentro de la ejecución del hilo, las cuáles se muestran a continuación [14].

Las propiedades a las que el hilo tiene acceso son:

Tabla 13. Propiedades de un hilo en la clase Thread

|  |  |
| --- | --- |
| PROPIEDADES | DEFINICIÓN |
| APARTMENTSTATE | Obtiene o establece el estado del apartamento de este subproceso (Obsoleto: Se ha dejado de utilizar por lo que su soporte ya no está funcionando) [14]. |
| CURRENTCULTURE | Obtiene o establece la referencia cultural del subproceso actual [14]. |
| CURRENTPRINCIPAL | [Obtiene](https://dotnet.microsoft.com/es-es/apps/desktop) o establece la entidad de seguridad actual del subproceso (para la seguridad basada en funciones) [14]. |
| CURRENTTHREAD | Obtiene el subproceso actualmente en ejecución [14]. |
| CURRENTUICULTURE | Obtiene o establece la referencia cultural actual utilizada por Resource Manager para buscar recursos específicos de la referencia cultural en tiempo de ejecución [14]. |
| EXECUTIONCONTEXT | Obtiene un objeto ExecutionContext que contiene información sobre los diversos contextos del subproceso actual [14]. |
| ISALIVE | Obtiene un valor que indica el estado de ejecución del subproceso actual [14]. |
| ISBACKGROUND | Obtiene o establece un valor que indica si un subproceso es o no un subproceso en segundo plano [14]. |
| ISTHREADPOOLTHREAD | Obtiene un valor que indica si un subproceso pertenece o no al grupo de subprocesos administrado [14]. |
| MANAGEDTHREADID | Obtiene un identificador único para el subproceso administrado actual [14]. |
| NAME | Obtiene o establece el nombre del subproceso [14]. |
| PRIORITY | Obtiene o establece un valor que indica la prioridad de programación de un subproceso [14]. |
| THREADSTATE | Obtiene un valor que contiene los estados del subproceso actual [14]. |

Los Métodos a los que el hilo tiene acceso son:

Tabla 14. Métodos de un hilo en la clase Thread

|  |  |
| --- | --- |
| MÉTODOS | DEFINICIÓN |
| ABORT() | Obsoleto. Genera una ThreadAbortException en el subproceso en el que se invoca, para comenzar el proceso de terminación del subproceso. Llamar a este método generalmente termina el hilo [14]. |
| ABORT(OBJECT) | Obsoleto. Genera una excepción ThreadAbortException en el subproceso en el que se invoca, para comenzar el proceso de terminación del subproceso y, al mismo tiempo, proporciona información de excepción sobre la terminación del subproceso. Llamar a este método generalmente termina el hilo [14]. |
| ALLOCATEDATASLOT() | Asigna una ranura de datos sin nombre en todos los subprocesos. Para un mejor rendimiento, utilice campos que estén marcados con el atributo ThreadStaticAttribute en su lugar [14]. |
| ALLOCATENAMEDDATASLOT(STRING) | Asigna una ranura de datos con nombre en todos los subprocesos. Para un mejor rendimiento, utilice campos que estén marcados con el atributo ThreadStaticAttribute en su lugar [14]. |
| BEGINCRITICALREGION() | Notifica a un host que la ejecución está a punto de ingresar a una región de código en la que los efectos de un aborto de subproceso o una excepción no controlada pueden poner en peligro otras tareas en el dominio de la aplicación [14]. |
| BEGINTHREADAFFINITY() | Notifica a un host que el código administrado está a punto de ejecutar instrucciones que dependen de la identidad del subproceso del sistema operativo físico actual [14]. |
| DISABLECOMOBJECTEAGERCLEANUP() | Desactiva la limpieza automática de contenedores a los que se puede llamar en tiempo de ejecución (RCW) para el subproceso actual [14]. |
| ENDCRITICALREGION() | Notifica a un host que la ejecución está a punto de ingresar a una región de código en la que los efectos de un aborto de subproceso o una excepción no controlada se limitan a la tarea actual [14]. |
| ENDTHREADAFFINITY() | Notifica a un host que el código administrado ha terminado de ejecutar instrucciones que dependen de la identidad del subproceso del sistema operativo físico actual [14]. |
| EQUALS(OBJECT) | Determina si el objeto especificado es igual al objeto actual. (Heredado de Objeto) [14]. |
| FINALIZE() | Garantiza que los recursos se liberen y se realicen otras operaciones de limpieza cuando el recolector de elementos no utilizados reclama el objeto Thread [14]. |
| FREENAMEDDATASLOT(STRING) | Elimina la asociación entre un nombre y un slot, para todos los subprocesos del proceso. Para un mejor rendimiento, utilice campos que estén marcados con el atributo ThreadStaticAttribute en su lugar [14]. |
| GETAPARTMENTSTATE() | Devuelve un valor de ApartmentState que indica el estado del apartamento [14]. |
| GETCOMPRESSEDSTACK() | Obsoleto. Devuelve un objeto CompressedStack que se puede usar para capturar la pila del subproceso actual [14]. |
| GETCURRENTPROCESSORID() | Obtiene un ID que se usa para indicar en qué procesador se está ejecutando el subproceso actual [14]. |
| GETDATA(LOCALDATASTORESLOT) | Recupera el valor de la ranura especificada en el subproceso actual, dentro del dominio actual del subproceso actual. Para un mejor rendimiento, utilice campos que estén marcados con el atributo ThreadStaticAttribute en su lugar [14]. |
| GETDOMAIN() | Devuelve el dominio actual en el que se ejecuta el subproceso actual [14]. |
| GETDOMAINID() | Devuelve un identificador de dominio de aplicación único [14]. |
| GETHASHCODE() | Devuelve un código hash para el subproceso actual [14]. |
| GETNAMEDDATASLOT(STRING) | Busca una ranura de datos con nombre. Para un mejor rendimiento, utilice campos que estén marcados con el atributo ThreadStaticAttribute en su lugar [14]. |
| GETTYPE() | Obtiene el Tipo de la instancia actual. (Heredado de Objeto) [14]. |
| INTERRUPT() | Interrumpe un subproceso que se encuentra en el estado de subproceso WaitSleepJoin [14]. |
| JOIN() | Bloquea el subproceso de llamada hasta que finaliza el subproceso representado por esta instancia, mientras continúa realizando el bombeo estándar de COM y SendMessage [14]. |
| JOIN(INT32) | Bloquea el subproceso de llamada hasta que finaliza el subproceso representado por esta instancia o transcurre el tiempo especificado, mientras continúa realizando el bombeo estándar de COM y SendMessage [14]. |
| JOIN(TIMESPAN) | Bloquea el subproceso de llamada hasta que finaliza el subproceso representado por esta instancia o transcurre el tiempo especificado, mientras continúa realizando el bombeo estándar de COM y SendMessage [14]. |
| MEMBERWISECLONE() | Crea una copia superficial del objeto actual. (Heredado de Objeto) [14]. |
| MEMORYBARRIER() | Sincroniza el acceso a la memoria de la siguiente manera: el procesador que ejecuta el subproceso actual no puede reordenar las instrucciones de tal manera que los accesos a la memoria antes de la llamada a MemoryBarrier() se ejecuten después de los accesos a la memoria que siguen a la llamada a MemoryBarrier() [14]. |
| RESETABORT() | Obsoleto. Cancela un Abort(Object) solicitado para el subproceso actual [14]. |
| RESUME() | Obsoleto. Reanuda un hilo que ha sido suspendido [14]. |
| SETAPARTMENTSTATE(APARTMENTSTATE) | Establece el estado de apartamento de un subproceso antes de que se inicie [14]. |
| SETCOMPRESSEDSTACK(COMPRESSEDSTACK) | Obsoleto. Aplica un CompressedStack capturado al subproceso actual [14]. |
| SETDATA(LOCALDATASTORESLOT, OBJECT) | Establece los datos en la ranura especificada en el subproceso que se está ejecutando actualmente, para el dominio actual de ese subproceso. Para un mejor rendimiento, utilice campos marcados con el atributo ThreadStatic Attribute en su lugar [14]. |
| SLEEP(INT32) | Suspende el subproceso actual durante el número especificado de milisegundos [14]. |
| SLEEP(TIMESPAN) | Suspende el subproceso actual durante la cantidad de tiempo especificada [14]. |
| SPINWAIT(INT32) | Hace que un subproceso espere el número de veces definido por el parámetro de iteraciones [14]. |
| START() | Hace que el sistema operativo cambie el estado de la instancia actual a En ejecución [14]. |
| START(OBJECT) | Hace que el sistema operativo cambie el estado de la instancia actual a En ejecución y, opcionalmente, proporciona un objeto que contiene datos para que los utilice el método que ejecuta el subproceso [14]. |
| SUSPEND() | Obsoleto. O suspende el subproceso, o si el subproceso ya está suspendido, no tiene efecto [14]. |
| TOSTRING() | Devuelve una cadena que representa el objeto actual. (Heredado de Objeto) [14]. |
| TRYSETAPARTMENTSTATE(APARTMENTSTATE) | Establece el estado de apartamento de un subproceso antes de que se inicie [14]. |
| UNSAFESTART() | Hace que el sistema operativo cambie el estado de la instancia actual a En ejecución [14]. |
| UNSAFESTART(OBJECT) | Hace que el sistema operativo cambie el estado de la instancia actual a En ejecución y, opcionalmente, proporciona un objeto que contiene datos para que los utilice el método que ejecuta el subproceso [14]. |
| VOLATILEREAD(DOUBLE) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(INT16) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(INT32) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(INT64) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(INTPTR) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(OBJECT) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(SBYTE) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(SINGLE) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(UINT16) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(UINT32) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(UINT64) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEREAD(UINTPTR) | Lee el valor de un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura después de este método en el código, el procesador no puede moverlo antes de este método [14]. |
| VOLATILEWRITE(BYTE,BYTE) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(DOUBLE,DOUBLE) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(INT16,INT16) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(INT32,INT32) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(INT64,INT64) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(INTPTR,INTPTR) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(OBJECT,OBJECT) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(SBYTE,SBYTE) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(SINGLE,SINGLE) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(UINT16,UINT16) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(UINT32,UINT32) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(UINT64,UINT64) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| VOLATILEWRITE(UINTPTR,UINTPTR) | Escribe un valor en un campo. En los sistemas que lo requieren, inserta una barrera de memoria que evita que el procesador reordene las operaciones de memoria de la siguiente manera: si aparece una lectura o escritura antes de este método en el código, el procesador no puede moverla después de este método [14]. |
| YIELD() | Hace que el subproceso de llamada ceda la ejecución a otro subproceso que está listo para ejecutarse en el procesador actual. El sistema operativo selecciona el subproceso al que ceder [14]. |

### **1.3.13. SINCRONIZACIÓN DE HILOS**

Para crear aplicaciones basadas en múltiples threads es necesario hacer que estos coordinen sus actividades de acceso a recursos y datos compartidos. A este mecanismo de coordinación de threads se le conoce comúnmente como sincronización [15].

La sincronización consiste en realizar las siguientes tareas:

* Hay que asegurar que los diferentes threads en ejecución accedan uno tras otro (en serie) a los datos o los recursos compartidos para evitar la corrupción de estos [15].
* Hay que asegurar que los threads se ejecutan en el momento oportuno, y que además eviten la sobrecarga del sistema por permanecer tanto tiempo en espera [15].

Para esto, Microsoft .NET Framework cuenta la clase System.Threading.Monitor . Con esta clase se logra que uno y solo un thread acceda a una región de código (está región de código puede comprender el acceso/modificación a un archivo, la conexión a un recurso de red [e.g., base de datos, servicio Web]). Precisamente el miembro estático de clase usado para la tarea anterior consiste en el método Enter [15].

A la región de código que mencionamos anteriormente, también se le conoce como región crítica. Esta región crítica debe ser bloqueada por una instancia de un tipo por referencia, inclusive el identificador de la instancia actual this. (Se recomiendan usar cualquier otra instancia como bloqueador en lugar de this.) Una vez que un thread alcanza esta área de código crítica, otro thread que intente hacer lo mismo pasará al estado WaitSleepJoin (cfr. [Obtener el Estado de un Thread](http://ortizol.blogspot.com/2014/07/receta-multithreading-en-csharp-no-1-5-obtener-el-estado-de-un-thread.html)) y será agregado a la cola de espera hasta que la región se libere por el objeto bloqueador con la invocación del método Monitor.Exit. Es recomendable encerrar la sección de código crítica en un bloque try y la invocación de Exit sobre el bloque finally [15].

Así:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 10. Ejemplo Monitor.Exit C#

Con el objeto locker especificamos la instancia que se encargara de bloquear la región de código crítica. Observemos el uso del método Enter; a este método se le pasa como argumento la referencia que bloquea la región de código. En el bloque finally se invoca al método Exit para liberar la sección crítica [15].

Por otra parte, para situaciones de sincronización más complejas; como por ejemplo para la activación de determinados threads de la cola de espera y el control sobre cantidades ingentes de threads que podrían sobrecargar la unidad central de procesamiento (CPU);  la clase Monitor cuenta con los siguientes métodos:

* **Wait :** Se encarga de liberar el bloqueo y el thread en la cola de espera [15].
* **Pulse :** Notifica a uno de los threads en la cola de espera que el estado del objeto bloqueante (i.e., locker) ha cambiado su estado. El thread pasa a la cola de preparados (ready queue) [15].
* **PulseAll :** Notifica a todos los threads en la cola de espera que el estado del objeto bloqueante ha cambiado su estado. Todos los threads pasan a la cola de preparados (ready queue) [15].

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 11. Sincronización de Hilos C#

### **1.3.14. EXCEPCIONES EN HILOS**

En Dotnet existen seis principales excepciones utilizadas para hilos y su correcto funcionamiento, el uso de excepciones brinda un mejor alcance a la experiencia del usuario por lo que en algunos casos su uso es obligatorio, pero en otros no lo amerita sin embargo es una buena práctica de programación utilizar excepciones para poder dar una mejor experiencia al usuario. El uso de una excepción es realizado a partir de (try-catch) visto anteriormente en la Figura 16 y Figura 17). Entre las excepciones de la clase Thread en Dotnet se tiene:

Tabla 15. Excepciones en Hilos

|  |  |
| --- | --- |
| EXCEPCIONES | DEFINICIÓN |
| THREADABORTEXCEPTION | public sealed class ThreadAbortException : SystemException  La excepción que se produce cuando se realiza una llamada al método Abort(Object). Esta clase no puede heredarse [16]. |
| THREADEXCEPTIONEVENTARGS | public class ThreadExceptionEventArgs : EventArgs  Proporciona datos para el evento ThreadException el cual se produce con una excepción de subproceso no atrapada [17]. |
| THREADEXCEPTIONEVENTHANDLER | public delegate void ThreadExceptionEventHandler(object sender, ThreadExceptionEventArgs e);  Representa el método que controlará el evento ThreadException de una aplicación [18]. |
| THREADINTERRUPTEDEXCEPTION | public class ThreadInterruptedException : SystemException  La excepción que se produce cuando se interrumpe un subproceso mientras está en estado de espera [19]. |
| THREADSTARTEXCEPTION | public sealed class ThreadStartException : SystemException  La excepción que se produce cuando se produce un error en un subproceso administrado después de que se haya iniciado el subproceso del sistema operativo subyacente, pero antes de que el subproceso esté listo para ejecutar el código de usuario [20]. |
| THREADSTATEEXCEPTION | La excepción que se genera cuando un subproceso se encuentra en un estado de subproceso no válido para la llamada al método [21]. |

### **1.3.15. INTERRUPCIONES Y BLOQUEOS**

Al llamar al método [Thread.Sleep](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep), el subproceso actual se bloquea inmediatamente durante el número de milisegundos o el intervalo de tiempo que pase al método, y el resto de su intervalo de tiempo se dedica a otro subproceso. Una vez que transcurre ese intervalo, el subproceso en suspensión vuelve a ejecutarse [22]. Un subproceso no puede llamar a [Thread.Sleep](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep) en otro subproceso. [Thread.Sleep](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep) es un método estático que siempre hace que el subproceso actual entre en modo de suspensión [22].

Al llamar a [Thread.Sleep](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.sleep) con un valor de [Timeout.Infinite](https://learn.microsoft.com/es-es/dotnet/api/system.threading.timeout.infinite), un subproceso entra en modo de suspensión hasta que lo interrumpe otro subproceso que llama al método [Thread.Interrupt](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt) en el subproceso suspendido, o hasta que una llamada a su método [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) lo termina. En el ejemplo siguiente se muestran ambos métodos para interrumpir un subproceso en suspensión [22].

Texto

Descripción generada automáticamente

Figura 12. Ejemplo de bloqueo con hilos en C#

Si una espera es administrada, tanto [Thread.Interrupt](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt) como [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) activan el subproceso inmediatamente. Si una espera es de tipo no administrado (por ejemplo, una llamada de invocación de plataforma a la función [WaitForSingleObject](https://learn.microsoft.com/es-es/windows/desktop/api/synchapi/nf-synchapi-waitforsingleobject) de Win32).

Ni [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) ni [Thread.Interrupt](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt) pueden tomar el control del subproceso hasta que este vuelva o llame a código administrado. En código administrado, el comportamiento es el siguiente [22]:

* [**Thread.Interrupt**](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.interrupt) activa un subproceso de cualquier tipo de espera que pueda haber y hace que se genere una [ThreadInterruptedException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadinterruptedexception) en el subproceso de destino [22].
* **Solo .NET Framework:** [Thread.Abort](https://learn.microsoft.com/es-es/dotnet/api/system.threading.thread.abort) activa un subproceso de cualquier tipo de espera en el que pueda estar y hace que se inicie una [ThreadAbortException](https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadabortexception) en el subproceso. Para obtener detalles, vea [Destrucción de subprocesos](https://learn.microsoft.com/es-es/dotnet/standard/threading/destroying-threads) [22].

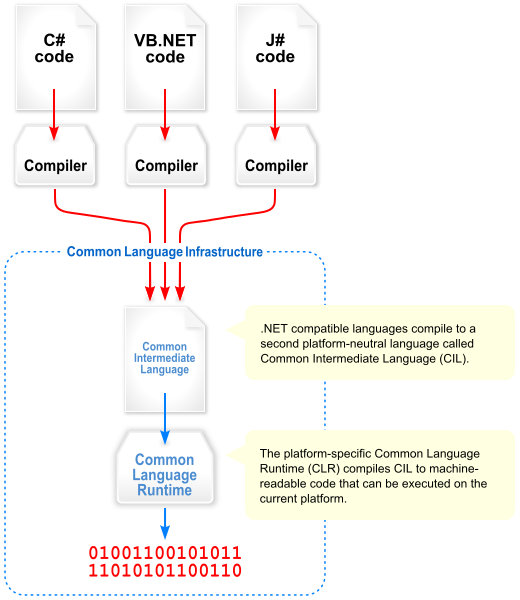


Figura 13.Interrupciones y Bloqueos

### **1.3.16. CONSTRUCTORES**

Para inicializar una nueva instancia de la clase Thread, se puede llamar a diferentes tipos de constructores, entre ellos [23]:

Tabla 16. Constructores de la clase Thread

|  |  |
| --- | --- |
| CONSTRUCTORES | DEFINICIÓN |
| THREAD(PARAMETERIZEDTHREADSTART) | Inicializa una nueva instancia de la clase Thread, especificando un delegado que permite pasar un objeto al subproceso cuando se inicia el subproceso [23]. |
| THREAD(THREADSTART) | Inicializa una nueva instancia de la clase Thread [23]. |
| THREAD(PARAMETERIZEDTHREADSTART,  INT32) | Inicializa una nueva instancia de la clase Thread, especificando un delegado que permite pasar un objeto al subproceso cuando se inicia el subproceso y especificando el tamaño máximo de pila para el subproceso [23]. |
| THREAD(THREADSTART,INT32) | Inicializa una nueva instancia de la clase Thread, especificando el tamaño de pila máximo para el hilo [23]. |

## FRAMEWORK

El Framework es una especie de plantilla, un esquema conceptual, que simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo con lo que se quiere realizar. A pesar de que su uso más común es en la informática, este concepto es también utilizado en el [Marketing](https://rockcontent.com/es/blog/marketing-2/). En ambas áreas facilita mucho el trabajo, ya sea del desarrollo de un software o de una estrategia de [mercadotecnia](https://rockcontent.com/es/blog/que-es-mercadotecnia/) [24].

## SISTEMAS MONOTAREA Y MULTITAREA

#### Multitareas

En esencia la multitarea nos permite ejecutar varios procesos a la vez; es decir, de forma concurrente y por tanto eso nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes. Evidentemente no podemos ejecutar infinitos procesos de forma concurrente ya que el hardware tiene sus limitaciones, pero raro es a día de hoy los ordenadores que no tengan más de un núcleo por tanto en un procesador con dos núcleos se podrían ejecutar dos procesos a la vez y así nuestro programa utilizaría al máximo los recursos hardware [25].

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

Figura 14. Representación de ejecución de 4 procesos

Sistemas Operativos multitareas: Windows 98, Windows me, Windows millenium, windows xp, Windows vista, Windows 7, Windows 8, Linux, Mac.

#### Monotarea

Los sistemas operativos monotarea son más primitivos y es todo lo contrario al visto anteriormente, es decir, solo pueden manejar un proceso en cada momento o que solo puede ejecutar las tareas de una en una. Sistemas Operativos monotarea: MS-DOS [25].

## PATRONES DE DISEÑO

Los patrones de diseño o design patterns, son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error [26].

Los patrones de diseño más utilizados se clasifican en tres categorías principales que son:

* Patrones creacionales
* Patrones estructurales
* Patrones de comportamiento

### **1.6.1. MODELO – VISTA – CONTROLADOR (MVC)**

MVC (Modelo-Vista-Controlador) es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Enfatiza una separación entre la lógica de negocios y su visualización. Esta "separación de preocupaciones" proporciona una mejor división del trabajo y una mejora de mantenimiento. Algunos otros patrones de diseño se basan en MVC, como MVVM (Modelo-Vista-modelo de vista), MVP (Modelo-Vista-Presentador) y MVW (Modelo-Vista-Whatever) [27].

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

1. Modelo: Maneja datos y lógica de negocios.
2. Vista: Se encarga del diseño y presentación.
3. Controlador: Enruta comandos a los modelos y vistas.

## NOMENCLATURA DE PROGRAMACIÓN

Si hablamos sobre nomenclatura de programación tenemos las dos más importantes que son LowerCammellCase y el UpperCammellCase que son dos estándares que se manejan para alrededor del mundo de la programación, pero primero tenemos que definir cada uno de estos y cuál será el que se usará en el presente proyecto [28].

**lowerCammellCase:** Cuando la primera letra de es una minúscula y las siguientes de cada palabra en mayúscula.

**UpperCammellCase:** Cuando la primera letra de cada una de las palabras es mayúscula.

Patrón de fondo, Rectángulo

Descripción generada automáticamente

Figura 15. Ejemplos de las nomenclaturas de programación

En el caso del presente proyecto se trabajará con lowerCammellCase ya que durante el transcurso de los presentes es con el que más se ha trabajado y a su vez este estándar de nomenclatura es lo que se usa con más frecuencia [28].

# PARTE PRÁCTICA

## CREACIÓN DE PRIMER PROYECTO DOTNET

Se abre el IDE de desarrollo Visual Studio 2022, se usará esta versión para la respectiva codificación y desarrollo del aplicativo ya que es la más actualizada en la fecha proporcionada.



Figura 16. Apertura del IDE Visual Studio 2022

Una vez realizado este proceso se abrirá un ayudante de la herramienta que permitirá seleccionar entre diferentes opciones que el IDE brinda, en este caso únicamente se seleccionará la opción de (Crear un proyecto). Un dato interesante es que aparte de las opciones que se muestran en la parte derecha también se muestran los últimos proyectos que se han realizado en la herramienta, esto permite tener un fácil acceso y manipulación al momento de gestionar el IDE.

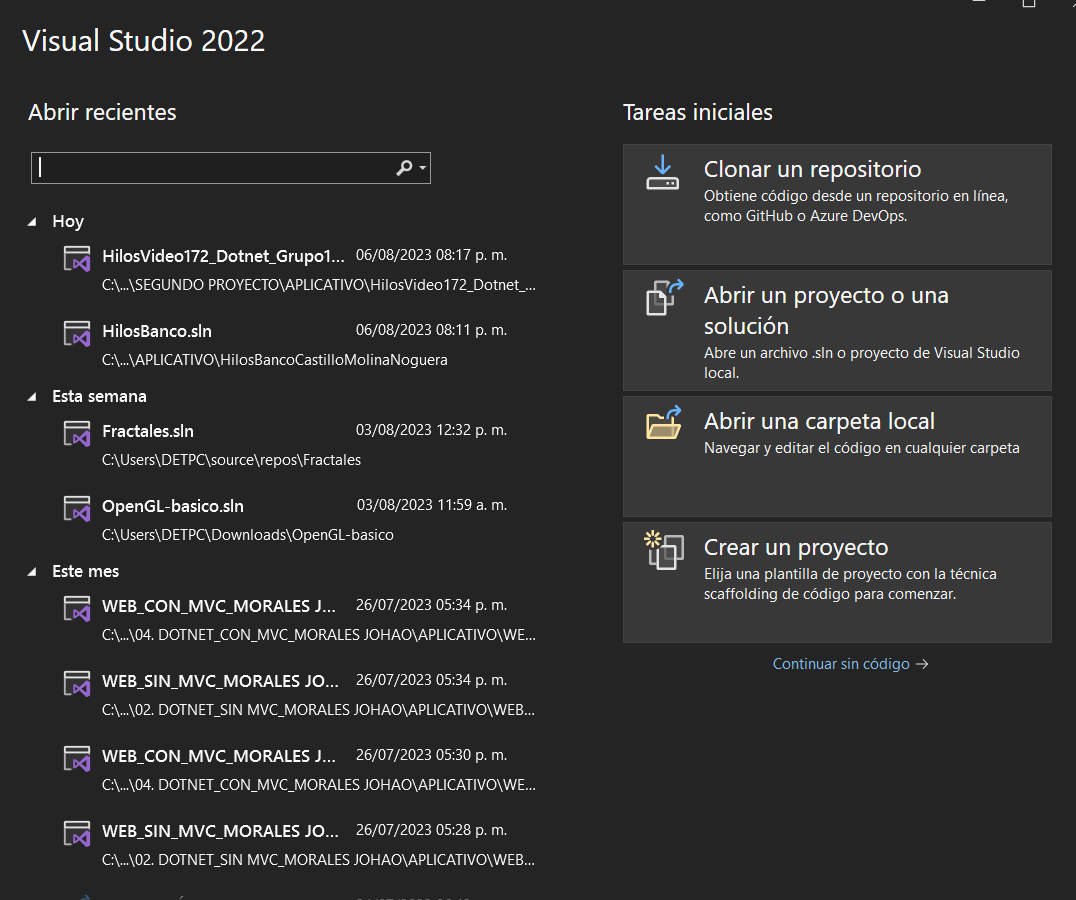


Figura 17. Ayudante Herramientas del IDE Visual Studio 2022

Una vez seleccionado sobre la creación de un nuevo proyecto se sobrepondrá una nueva ventana la cuál contendrá el tipo de aplicación que se va a construir, en este caso se realizará la aplicación de tipo Consola por lo cuál se seleccionará (Aplicación de consola).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 18. Creación de aplicación tipo consola IDE Visual Studio 2022

Una vez seleccionado el tipo de proyecto que se va a realizar se oprime sobre el botón siguiente -> en este caso se obtendrá una nueva ventana emergente en la cuál se deberá de colocar el nombre del proyecto, para este caso el nombre asignado será (HiloVideos171\_Dotnet\_Grupo1).

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 19. Nombramiento de proyecto IDE Visual Studio 2022

Al presionar sobre siguiente con la siguiente configuración, se obtendrá una nueva ventana emergente la cual presentará el uso de un Framework de desarrollo .Net 6.0, para el presente proyecto no se utilizará el uso de este Framework por lo que no se lo seleccionará, finalmente se oprimirá el botón ->Crear para poder generar así el nuevo proyecto de consola en el IDE Visual Studio 2022.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 20. Selección de Framework IDE Visual Studio 2022

Finalmente se ejecutará Visual Studio 2022 junto con el proyecto de tipo consola que se ha creado, por lo tanto se obtendrá una vista de un aplicativo similar a la siguiente.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 21. Primera Vista IDE Visual Studio 2022

## CREACIÓN DE PAQUETES EN PRIMER PROYECTO DOTNET

Para la generación de paquetes se debe dar click derecho sobre el proyecto que se generó, seguido de -> Agregar -> Nueva Carpeta.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 22. Agregación de una nueva carpeta proyecto Dotnet

Al generar la nueva carpeta se podrá dar un nombramiento a la misma en este caso el nombre para la carpeta del proyecto será específicamente (ec.edu.monster.modelo), seguido siguiendo el paso anterior se crearán dos carpetas adicionales las cuáles serán (ec.edu.monster.vista) y (ec.edu.monster.controlador), conteniendo una estructura similar a la siguiente.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 23. Creación de paquetes MVC en Dotnet

## CREACIÓN DE ARCHIVOS .CS EN PRIMER PROYECTO DOTNET

### **2.3.1 CODIFICACIÓN DEL ARCHIVO HILO.CS**

Se realizará la codificación respectiva para una nueva clase denominada (Hilo.cs), para generar esta clase se realizará Click Derecho sobre el paquete generado anteriormente (ec.edu.monster.modelo) seguido de ->Agregar -> Clase.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 24. Agregación de nueva clase Dotnet

Aparecerá una nueva ventana emergente en la cual se podrá seleccionar si efectivamente el componente de elementos a utilizar será el elemento Clase, seguido se deberá de dar un nombre a la misma, en este caso el nombre de la clase a utilizar será Hilo.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 25. Nombramiento de clase Hilo.cs Dotnet

Finalmente se obtendrá una clase similar a la proporcionada, siendo esta la clase base para el desarrollo del proyecto.

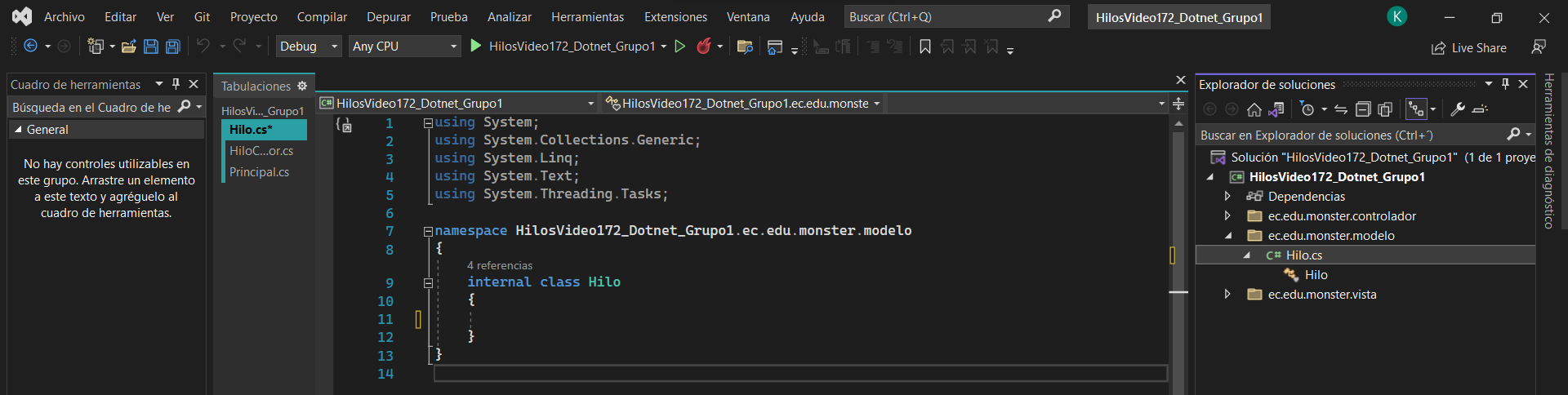


Figura 26. Creación del proyecto Hilo.cs Dotnet

Finalmente se realizará la codificación del archivo Hilo.cs, el cuál contendrá los siguientes valores, además se deberán de crear getters y setters respectivos para los valores dados.

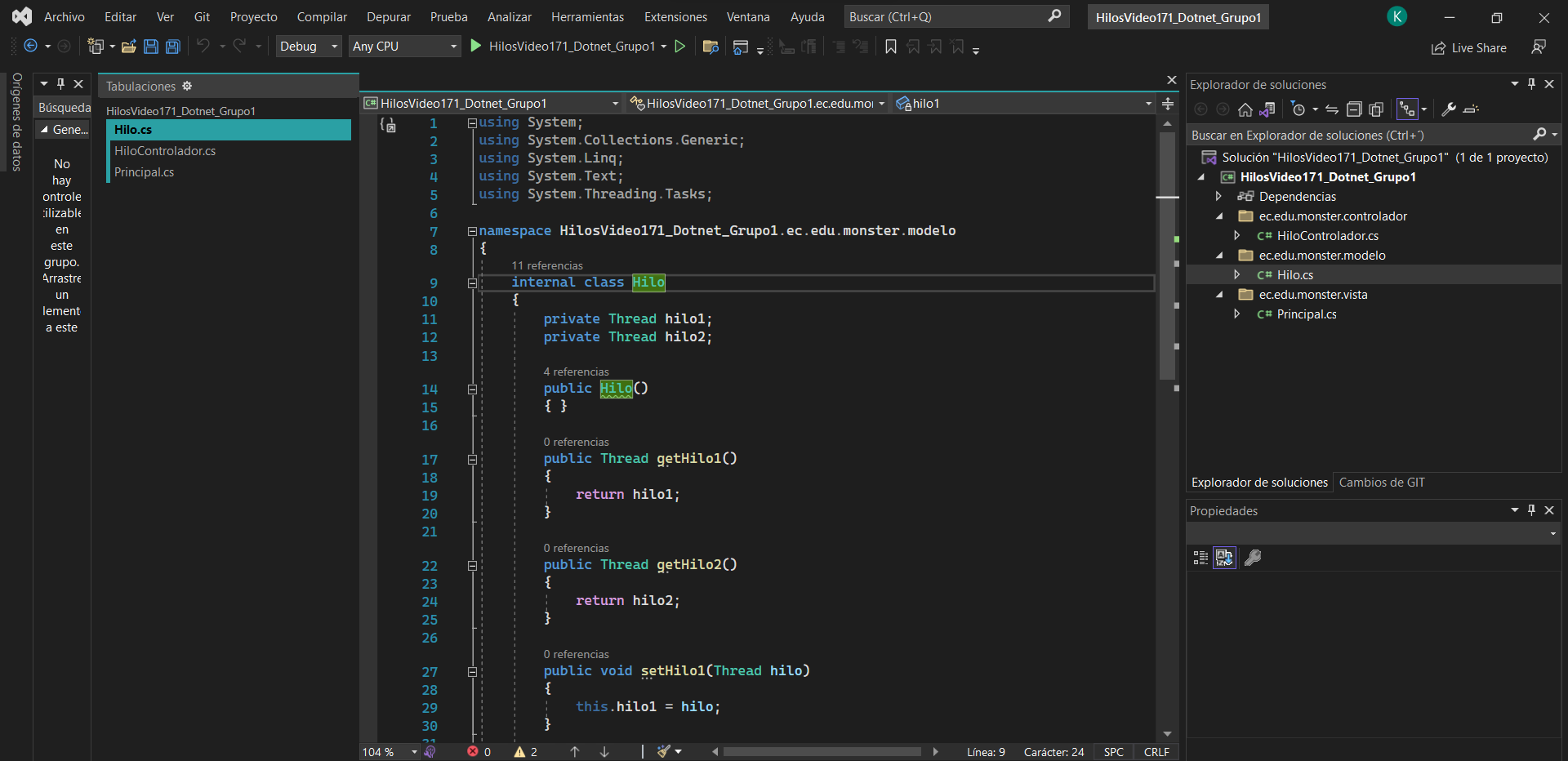


Figura 27. Codificación del archivo Hilo.cs

Tabla 17. Codificación del archivo Hilo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HilosVideo171\_Dotnet\_Grupo1.ec.edu.monster.modelo

{

internal class Hilo

{

private Thread hilo1;

private Thread hilo2;

public Hilo()

{ }

public Thread getHilo1()

{

return hilo1;

}

public Thread getHilo2()

{

return hilo2;

}

public void setHilo1(Thread hilo)

{

this.hilo1 = hilo;

}

public void setHilo2(Thread hilo)

{

this.hilo2 = hilo;

}

}

}

### **2.3.2 CODIFICACIÓN DEL ARCHIVO HILOCONTROLADOR.CS**

Se realizará la construcción del archivo HiloControlador.cs, para esto se deberá de dar click derecho sobre el paquete (ec.edu.monster.controlador) creado anteriormente seguido de -> Agregar -> Clase…

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 28. Agregación de nueva clase Dotnet

Una vez realizado este proceso aparecerá una nueva ventana emergente en la cual se confirmará la creación del elemento Clase, además del nombramiento de la misma, en este caso la clase será denominada (HiloControlador.cs).

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 29. Nombramiento de la clase HiloControlador.cs

Se obtendrá una clase base para la dada, la cuál deberá tener una similitud a la siguiente.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 30. Apertura básica de HiloControlador.cs

Se deberá realizar una respectiva codificación en la clase HiloControlador.cs en donde se obtendrán funciones respectivas para el manejo de hilos, su respectiva codificación será la siguiente.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 31. Codificación del archivo HiloControlador.cs

Tabla 18. Codificación del archivo HiloControlador.cs

using HilosVideo171\_Dotnet\_Grupo1.ec.edu.monster.modelo;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HilosVideo171\_Dotnet\_Grupo1.ec.edu.monster.controlador

{

internal class HiloControlador

{

private Hilo hilo1;

private Hilo hilo2;

public HiloControlador(Hilo hilo1, Hilo hilo2)

{

this.hilo1 = hilo1;

this.hilo2 = hilo2;

}

public void MostrarHiloEjecucion1()

{

for (int i = 0; i < 15; i++)

{

Console.WriteLine("Ejecutando Hilo Thread ==> 0");

hilo1 = new Hilo();

Thread.Sleep(500);

}

}

public void MostrarHiloEjecucion2()

{

for (int i = 0; i < 10; i++)

{

Console.WriteLine("Escribiendo Hilo Thread ==> 1");

hilo2 = new Hilo();

Thread.Sleep(500);

}

}

}

}

### **2.3.3 CODIFICACIÓN DEL ARCHIVO PRINCIPAL.CS**

Se realizará la construcción del archivo Principal.cs, para esto se deberá de dar click derecho sobre el paquete (ec.edu.monster.vista) creado anteriormente seguido de -> Agregar -> Clase …

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 32. Creación de nueva clase Dotnet

Una vez realizado este proceso se obtendrá una pantalla emergente en la cuál se verificará el componente seleccionado anteriormente, se dará un nombre para la clase en este caso Principal.cs.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 33. Selección y nombramiento de componente clase Dotnet

Se generará una clase base para la clase Principal.cs, seguido se deberá dar la siguiente codificación, sin embargo se puede notar un error en el cuál aparece que solo una unidad de compilación puede tener instrucciones de nivel superior.

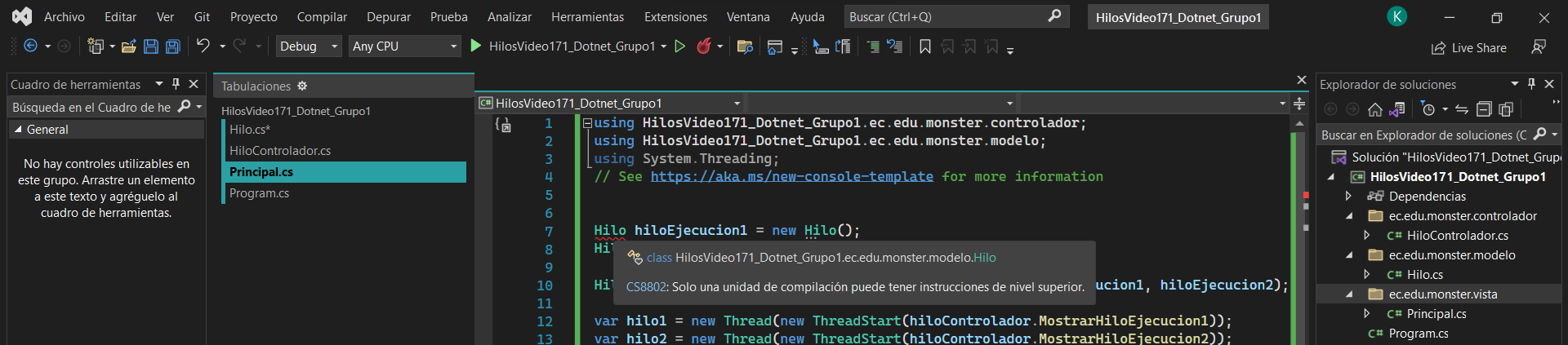


Figura 34. Codificación del archivo Principal.cs

Para realizar la correcta codificación se deberá de eliminar el archivo creado por defecto para la ejecución del proyecto denominado Program.cs, esto se debe a que se contiene mas de un archivo Main invocado para la ejecución.



Figura 35. Eliminación del archivo Program.cs

Al regresar al archivo Principal.cs se verificará que el archivo no contiene errores por lo que su codificación ha sido realizada de manera exitosa, se invocan a los métodos creados en (ec.edu.monster.controlador) y se realiza una correcta inicialización y codificación del archivo obteniendo como resultado algo similar a esta codificación.

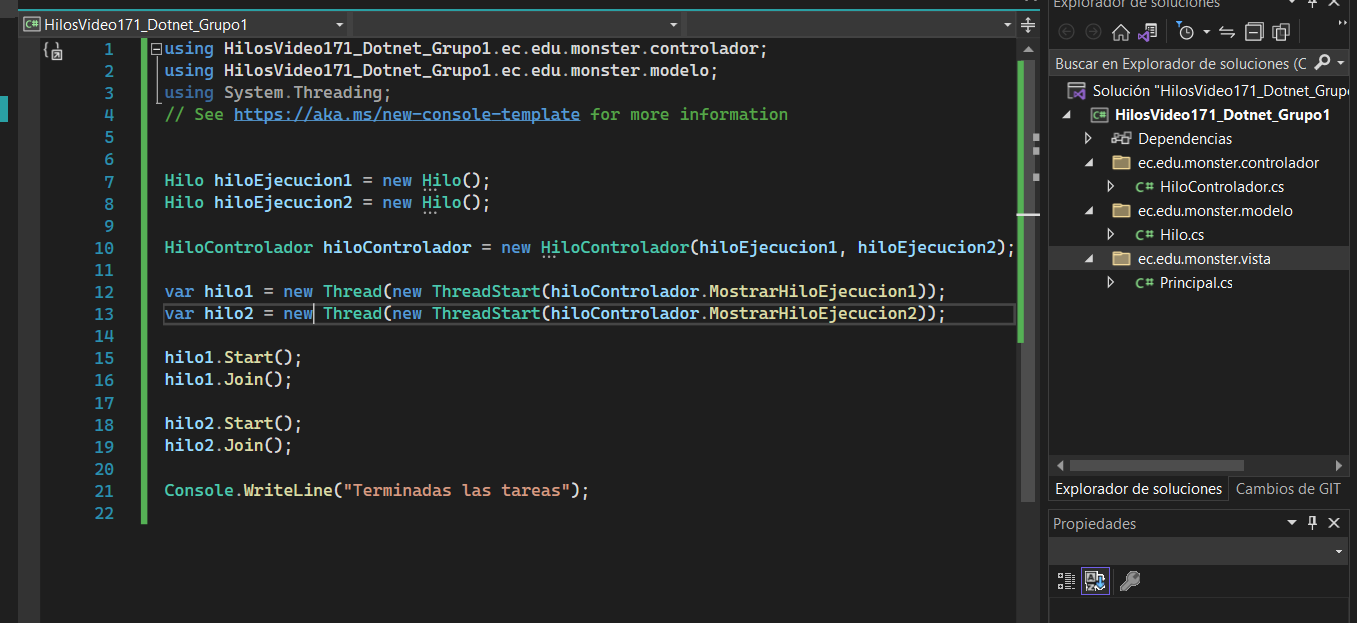


Figura 36. Codificación del archivo Principal.cs

Tabla 19. Codificación del archivo Principal.cs

using HilosVideo171\_Dotnet\_Grupo1.ec.edu.monster.controlador;

using HilosVideo171\_Dotnet\_Grupo1.ec.edu.monster.modelo;

using System.Threading;

// See https://aka.ms/new-console-template for more information

Hilo hiloEjecucion1 = new Hilo();

Hilo hiloEjecucion2 = new Hilo();

HiloControlador hiloControlador = new HiloControlador(hiloEjecucion1, hiloEjecucion2);

var hilo1 = new Thread(new ThreadStart(hiloControlador.MostrarHiloEjecucion1));

var hilo2 = new Thread(new ThreadStart(hiloControlador.MostrarHiloEjecucion2));

hilo1.Start();

hilo1.Join();

hilo2.Start();

hilo2.Join();

Console.WriteLine("Terminadas las tareas");

## CREACIÓN DE SEGUNDO PROYECTO DOTNET

Se abre el IDE de desarrollo Visual Studio 2022, se usará esta versión para la respectiva codificación y desarrollo del aplicativo ya que es la más actualizada en la fecha proporcionada.



Figura 37. Apertura del IDE Visual Studio 2022

Una vez realizado este proceso se abrirá un ayudante de la herramienta que permitirá seleccionar entre diferentes opciones que el IDE brinda, en este caso únicamente se seleccionará la opción de (Crear un proyecto). Un dato interesante es que aparte de las opciones que se muestran en la parte derecha también se muestran los últimos proyectos que se han realizado en la herramienta, esto permite tener un fácil acceso y manipulación al momento de gestionar el IDE.

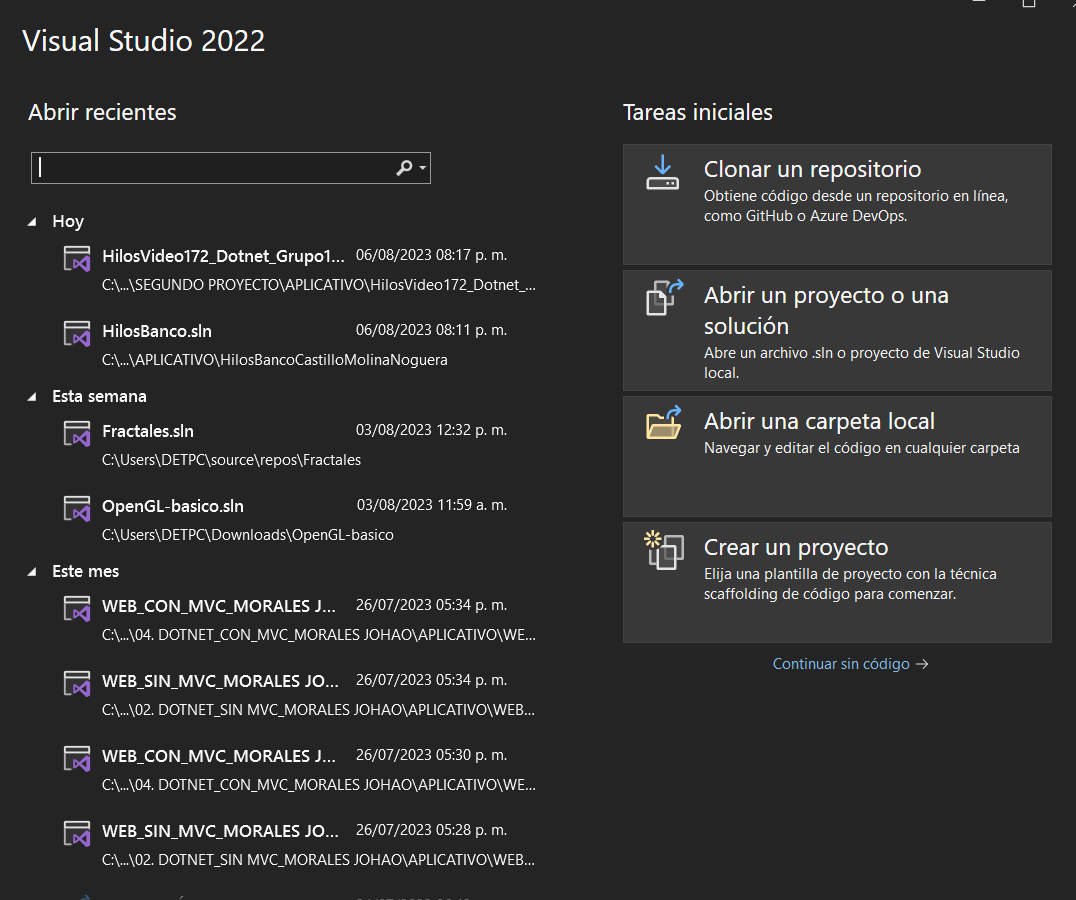


Figura 38. Ayudante Herramientas del IDE Visual Studio 2022

Una vez seleccionado sobre la creación de un nuevo proyecto se sobrepondrá una nueva ventana la cuál contendrá el tipo de aplicación que se va a construir, en este caso se realizará la aplicación de tipo Consola por lo cuál se seleccionará (Aplicación de consola).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 39. Creación de aplicación tipo consola IDE Visual Studio 2022

Una vez seleccionado el tipo de proyecto que se va a realizar se oprime sobre el botón siguiente -> en este caso se obtendrá una nueva ventana emergente en la cuál se deberá de colocar el nombre del proyecto, para este caso el nombre asignado será (HiloVideos172\_Dotnet\_Grupo1).

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 40. Nombramiento del proyecto Dotnet

Al presionar sobre siguiente con la siguiente configuración, se obtendrá una nueva ventana emergente la cual presentará el uso de un Framework de desarrollo .Net 6.0, para el presente proyecto no se utilizará el uso de este Framework por lo que no se lo seleccionará, finalmente se oprimirá el botón ->Crear para poder generar así el nuevo proyecto de consola en el IDE Visual Studio 2022.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 41. Selección de Framework IDE Visual Studio 2022

Finalmente se ejecutará Visual Studio 2022 junto con el proyecto de tipo consola que se ha creado, por lo tanto se obtendrá una vista de un aplicativo similar a la siguiente.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Figura 42. Generación proyecto base Dotnet

## CREACIÓN DE PAQUETES EN SEGUNDO PROYECTO DOTNET

Para la generación de paquetes se debe dar click derecho sobre el proyecto que se generó, seguido de -> Agregar -> Nueva Carpeta.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 43. Agregación de paquetes en el proyecto Dotnet

Al generar la nueva carpeta se podrá dar un nombramiento a la misma en este caso el nombre para la carpeta del proyecto será específicamente (ec.edu.monster.modelo), seguido siguiendo el paso anterior se crearán dos carpetas adicionales las cuáles serán (ec.edu.monster.vista) y (ec.edu.monster.controlador), conteniendo una estructura similar a la siguiente.

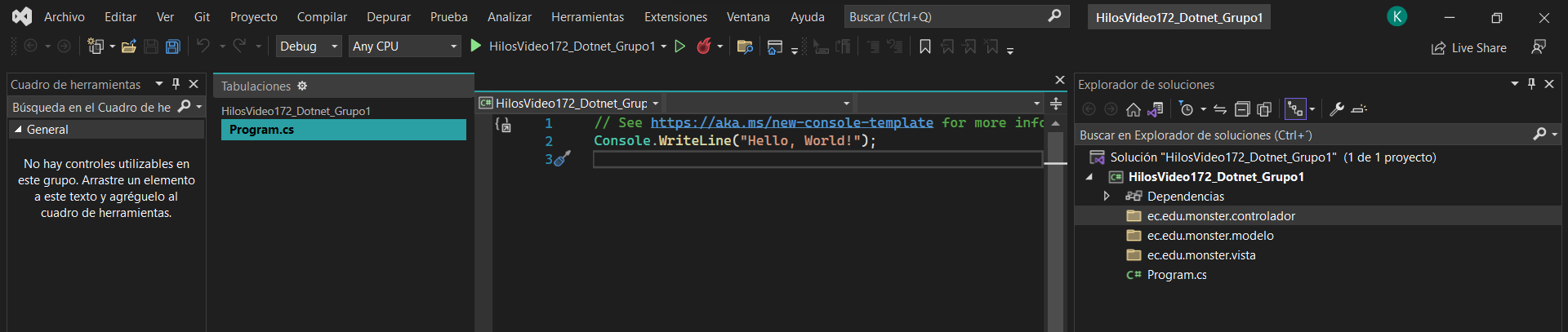


Figura 44. Agregación de paquetes MVC en Dotnet

## CREACIÓN DE ARCHIVOS .CS EN SEGUNDO PROYECTO DOTNET

### **2.6.1 CODIFICACIÓN DEL ARCHIVO HILO.CS**

Se realizará la codificación respectiva para una nueva clase denominada (Hilo.cs), para generar esta clase se realizará Click Derecho sobre el paquete generado anteriormente (ec.edu.monster.modelo) seguido de ->Agregar -> Clase.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 45. Agregación de nueva clase Dotnet

Aparecerá una nueva ventana emergente en la cual se podrá seleccionar si efectivamente el componente de elementos a utilizar será el elemento Clase, seguido se deberá de dar un nombre a la misma, en este caso el nombre de la clase a utilizar será Hilo.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 46. Nombramiento de clase Hilo.cs Dotnet

Finalmente se obtendrá una clase similar a la proporcionada, siendo esta la clase base para el desarrollo del proyecto.

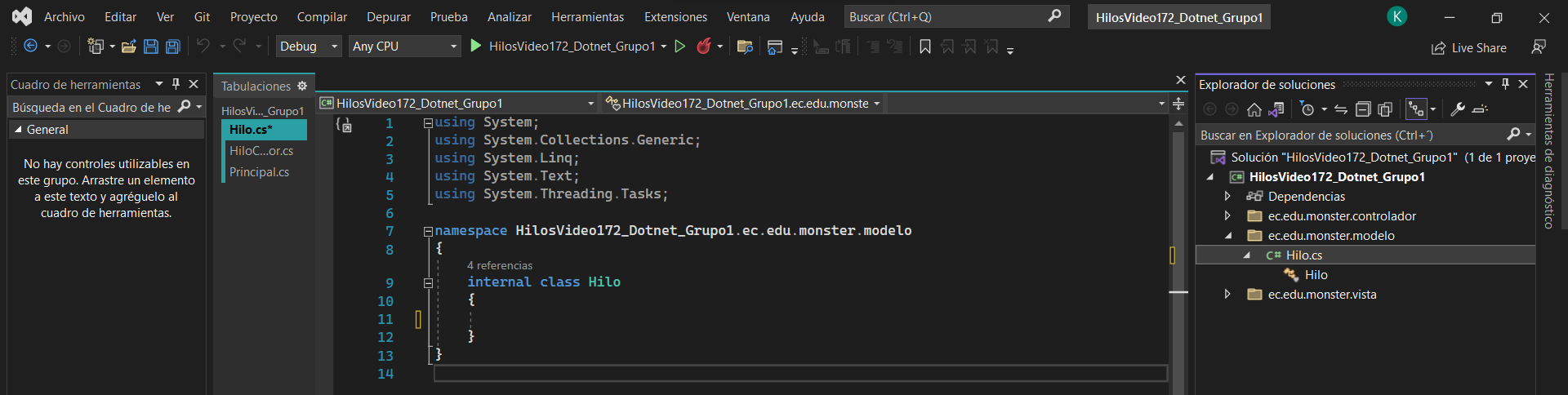


Figura 47. Creación del proyecto Hilo.cs Dotnet

Finalmente se realizará la codificación del archivo Hilo.cs, el cuál contendrá los siguientes valores, además se deberán de crear getters y setters respectivos para los valores dados.El archivo contendrá además de getters y setters un constructor que se mantendrá como vacío, el uso de este constructor será utilizado mediante la clase controlador.

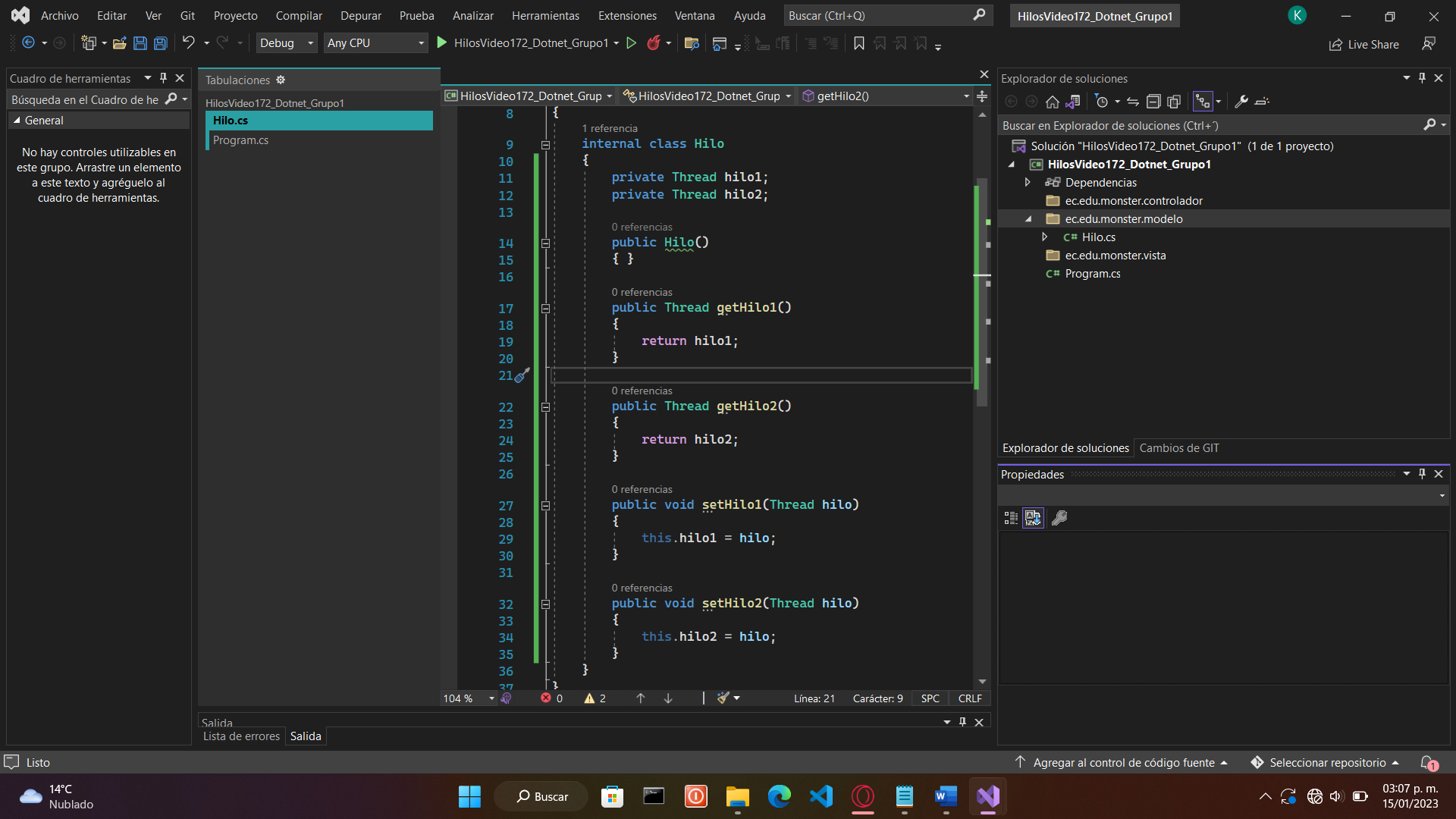


Figura 48. Codificación del archivo Hilo.cs

Tabla 20. Codificación del archivo Hilo.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HilosVideo172\_Dotnet\_Grupo1.ec.edu.monster.modelo

{

internal class Hilo

{

private Thread hilo1;

private Thread hilo2;

public Hilo()

{ }

public Thread getHilo1()

{

return hilo1;

}

public Thread getHilo2()

{

return hilo2;

}

public void setHilo1(Thread hilo)

{

this.hilo1 = hilo;

}

public void setHilo2(Thread hilo)

{

this.hilo2 = hilo;

}

}

}

### **2.6.2 CODIFICACIÓN DEL ARCHIVO HILOCONTROLADOR.CS**

Se realizará la construcción del archivo HiloControlador.cs, para esto se deberá de dar click derecho sobre el paquete (ec.edu.monster.controlador) creado anteriormente seguido de -> Agregar -> Clase…

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 49. Agregación de nueva clase Dotnet

Una vez realizado este proceso aparecerá una nueva ventana emergente en la cual se confirmará la creación del elemento Clase, además del nombramiento de la misma, en este caso la clase será denominada (HiloControlador.cs).

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 50. Nombramiento de la clase HiloControlador.cs

Se obtendrá una clase base para la dada, la cuál deberá tener una similitud a la siguiente.

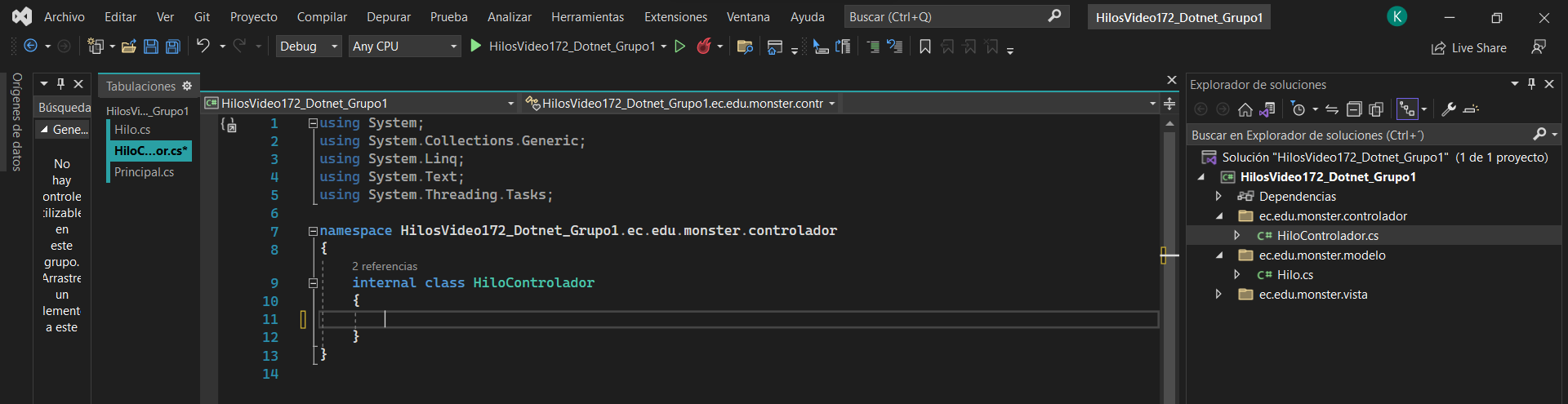
****

Figura 51. Generación de HiloControlador.cs

Se deberá realizar una respectiva codificación en la clase HiloControlador.cs en donde se obtendrán funciones respectivas para el manejo de hilos, su respectiva codificación será la siguiente.

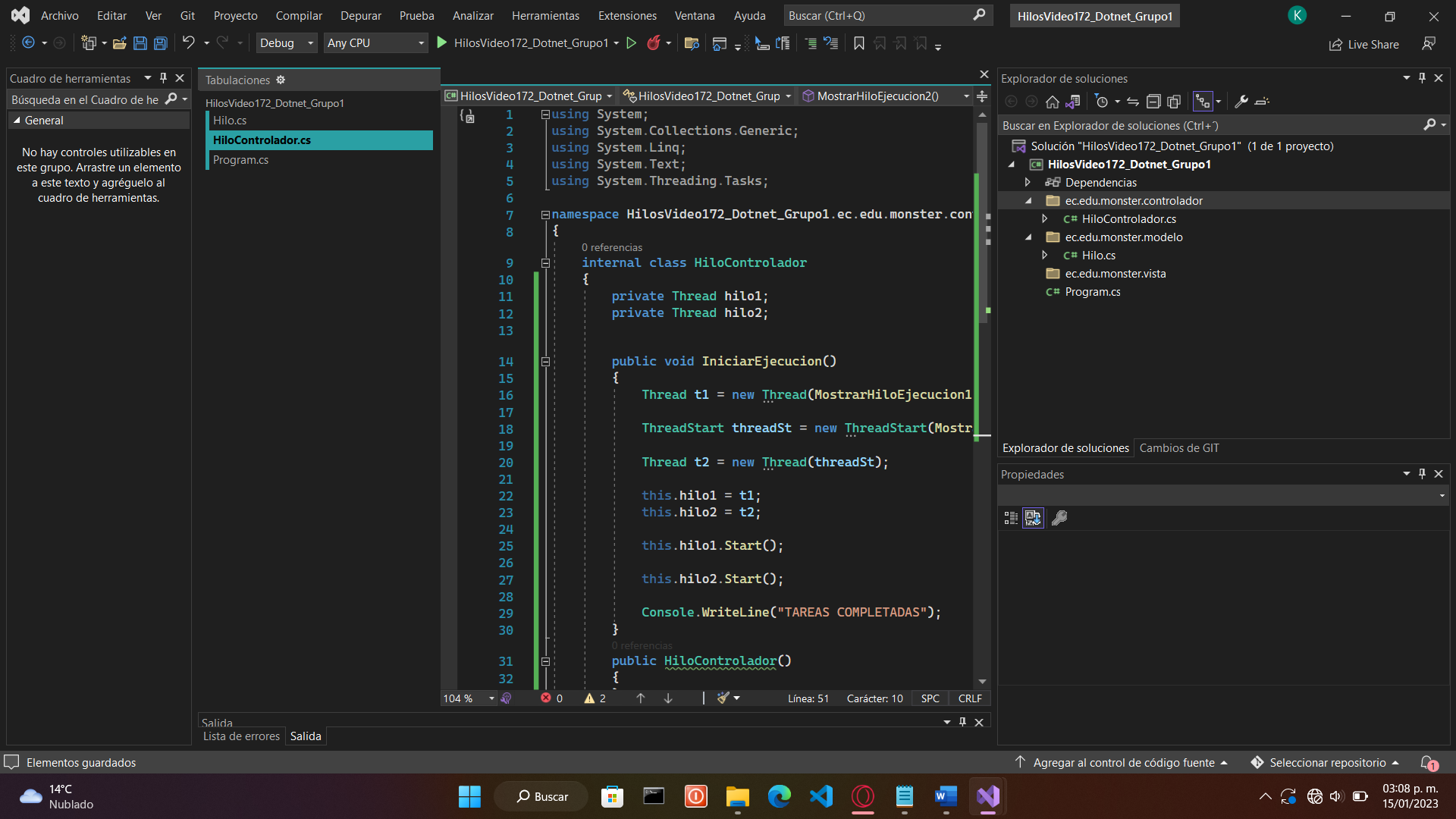


Figura 52. Codificación del archivo HiloControlador.cs

Tabla 21. Codificación del archivo HiloControlador.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HilosVideo172\_Dotnet\_Grupo1.ec.edu.monster.controlador{

internal class HiloControlador{

private Thread hilo1;

private Thread hilo2;

public void IniciarEjecucion(){

Thread t1 = new Thread(MostrarHiloEjecucion1);

ThreadStart threadSt = new ThreadStart(MostrarHiloEjecucion2);

Thread t2 = new Thread(threadSt);

this.hilo1 = t1;

this.hilo2 = t2;

this.hilo1.Start();

this.hilo2.Start();

Console.WriteLine("TAREAS COMPLETADAS");

}

public HiloControlador()

{}

public void MostrarHiloEjecucion1()

{

hilo2.Join();

for (int i = 0; i < 15; i++)

{

Console.WriteLine("Ejecutando Hilo Thread ==> 1");

Thread.Sleep(500);

}

}

public void MostrarHiloEjecucion2()

{

for (int i = 0; i < 15; i++)

{

Console.WriteLine("Escribiendo Hilo Thread ==> 0");

Thread.Sleep(500);

}

}

}

}

### **2.6.3 CODIFICACIÓN DEL ARCHIVO PRINCIPAL.CS**

Se realizará la construcción del archivo Principal.cs, para esto se deberá de dar click derecho sobre el paquete (ec.edu.monster.vista) creado anteriormente seguido de -> Agregar -> Clase …

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 53. Creación de nueva clase Dotnet

Una vez realizado este proceso se obtendrá una pantalla emergente en la cuál se verificará el componente seleccionado anteriormente, se dará un nombre para la clase en este caso Principal.cs.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 54. Selección y nombramiento de componente clase Dotnet

Se generará una clase base para la clase Principal.cs, seguido se deberá dar la siguiente codificación, sin embargo se puede notar un error en el cuál aparece que solo una unidad de compilación puede tener instrucciones de nivel superior.

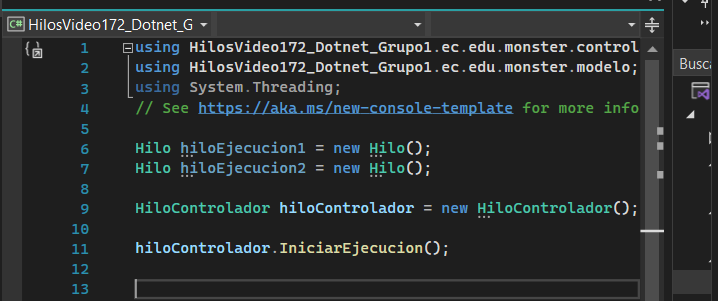
****

Figura 55. Codificación del archivo Principal.cs

Para realizar la correcta codificación se deberá de eliminar el archivo creado por defecto para la ejecución del proyecto denominado Program.cs, esto se debe a que se contiene mas de un archivo Main invocado para la ejecución.



Figura 56. Eliminación del archivo Program.cs

Al regresar al archivo Principal.cs se verificará que el archivo no contiene errores por lo que su codificación ha sido realizada de manera exitosa, se invocan a los métodos creados en (ec.edu.monster.controlador) y se realiza una correcta inicialización y codificación del archivo obteniendo como resultado algo similar a esta codificación.

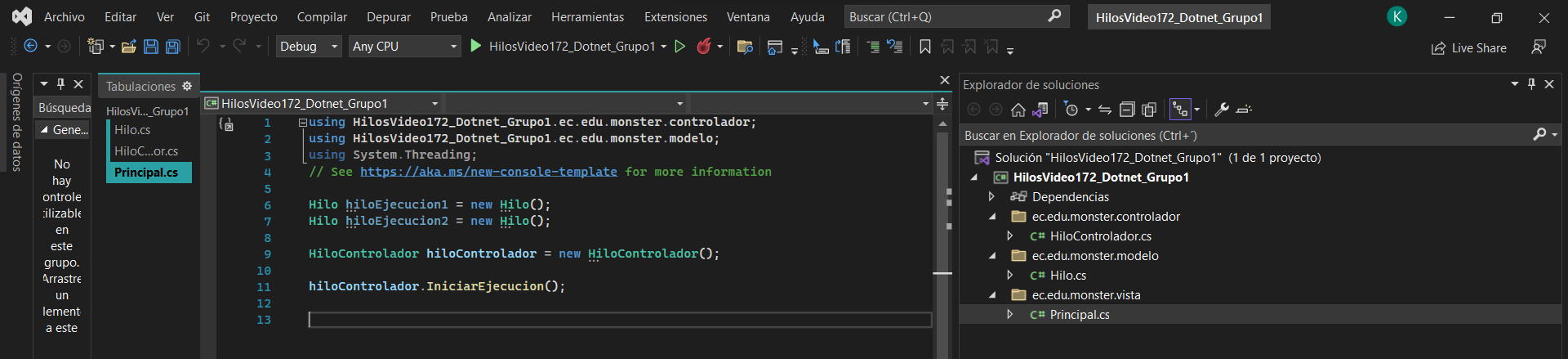


Figura 57. Codificación exitosa Principal.cs

Tabla 22. Codificación del archivo Principal.cs

using HilosVideo172\_Dotnet\_Grupo1.ec.edu.monster.controlador;

using HilosVideo172\_Dotnet\_Grupo1.ec.edu.monster.modelo;

using System.Threading;

// See https://aka.ms/new-console-template for more information

Hilo hiloEjecucion1 = new Hilo();

Hilo hiloEjecucion2 = new Hilo();

HiloControlador hiloControlador = new HiloControlador();

hiloControlador.IniciarEjecucion();

## FUNCIONALIDAD

### **ESTRUCTURA DE LAS APLICACIÓNES**

#### **2.7.1.1 PRIMER PROYECTO DOTNET HILOSVIDEO171\_DOTNET\_GRUPO1**

E proyecto dará la siguiente vista de la aplicación dotnet de tipo consola.

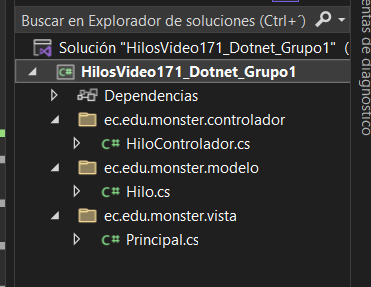


Figura 58. Estructura del primer proyecto Dotnet HilosVideo171\_Dotnet\_Grupo1

#### **2.7.1.2 SEGUNDO PROYECTO DOTNET HILOSVIDEO172\_DOTNET\_GRUPO1**

El proyecto dará la siguiente vista de la aplicación dotnet de tipo consola.

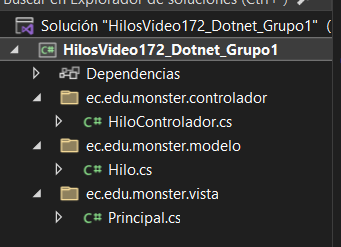


Figura 59. Estructura del segundo proyecto Dotnet HilosVideo172\_Dotnet\_Grupo1

## DESEMPEÑO DE LAS APLICACIÓNES

### **2.8.1 PRIMER PROYECTO DOTNET HILOSVIDEO171\_DOTNET\_GRUPO1**

Se ejecutará el aplicativo mediante el botón situado en la parte superior con símbolo de Play (Nombre del proyecto).



Figura 60. Ejecución de HilosVideo171\_Dotnet\_Grupo1

Una vez presionado sobre el botón se realizará una serie de impresiones por consola mostrando que primero se ejecutarán los hilos y finalmente las tareas siguientes.

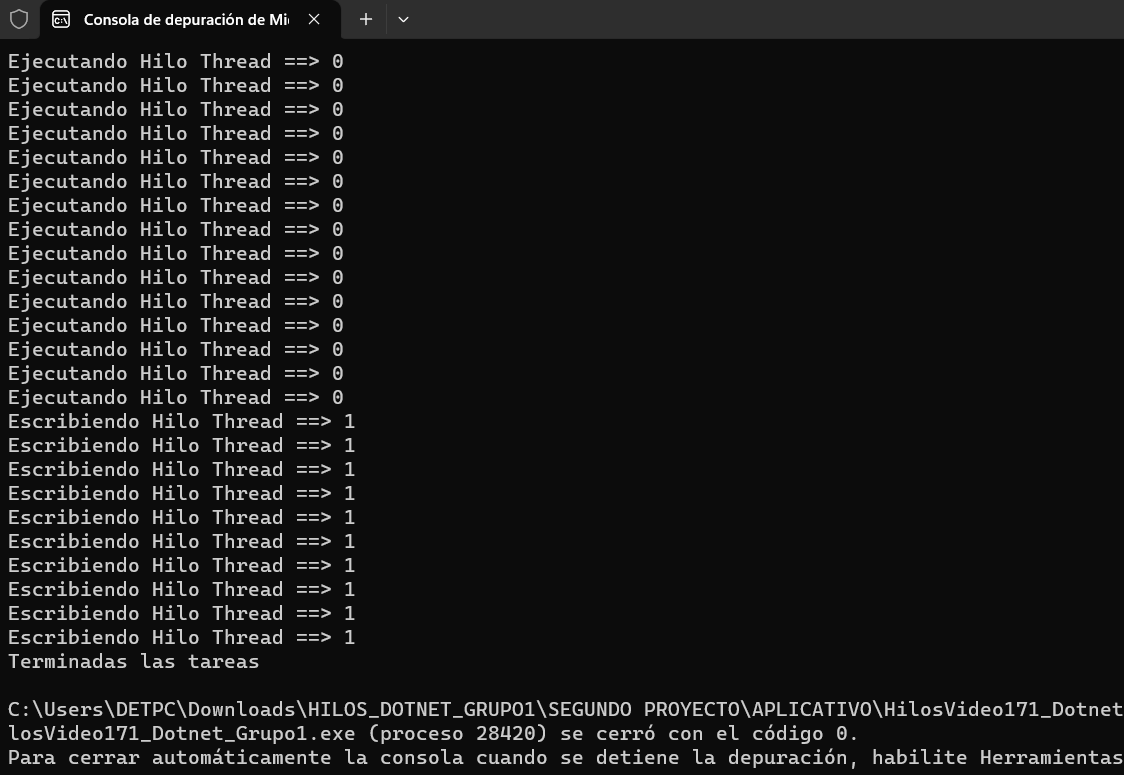


Figura 61. Ejecución en consola de HilosVideo171\_Dotnet\_Grupo1

### **2.8.2 SEGUNDO PROYECTO DOTNET HILOSVIDEO172\_DOTNET\_GRUPO1**

Se ejecutará el aplicativo mediante el botón situado en la parte superior con símbolo de Play (Nombre del proyecto).



Figura 62. Botón para ejecución HilosVideo172\_Dotnet\_Grupo1

Una vez presionado sobre el botón se realizará una serie de impresiones por consola mostrando que primero se ejecutarán las tareas respectivas seguidos de los hilos de ejecución.

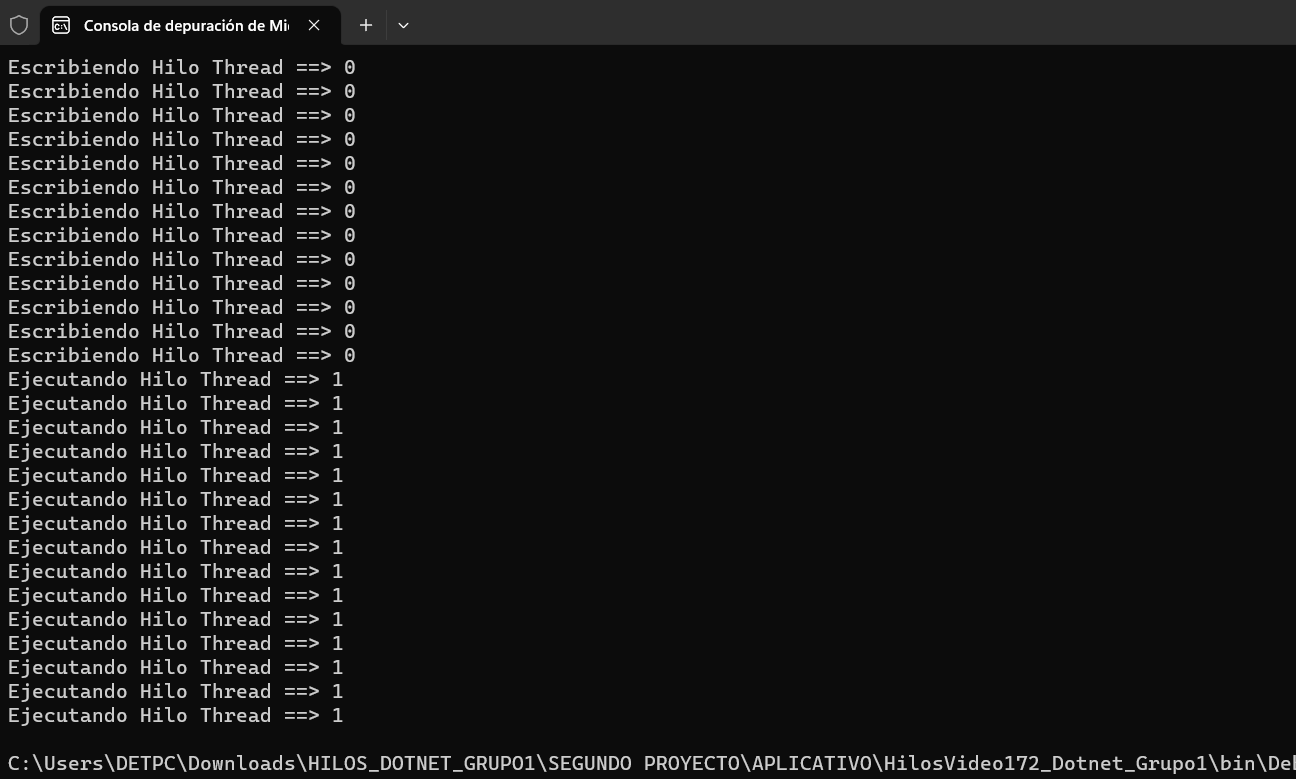


Figura 63. Ejecución en consola de HilosVideo172\_Dotnet\_Grupo1

# CONCLUSIONES

* En este proyecto, hemos explorado y aplicado el concepto de hilos tanto en .NET como en Java. A través de la práctica, hemos comprendido cómo manejarlos de manera efectiva. Inicialmente, partimos de una aplicación con un único hilo de ejecución y luego avanzamos hacia una implementación más compleja con múltiples hilos independientes. Esto nos permitió experimentar con la capacidad de generar mono tareas y multitareas en las aplicaciones.
* En el proceso, hemos aprendido sobre la versatilidad que los hilos aportan a las aplicaciones desarrolladas en ambos lenguajes. Observamos cómo una aplicación simple puede mejorar significativamente su funcionamiento y rendimiento al emplear hilos para desempeñar diversas tareas. Este aprendizaje reforzó la importancia de utilizar hilos en el desarrollo de aplicaciones .NET y cómo pueden potenciar la eficiencia y el rendimiento de dichas aplicaciones, especialmente cuando se combinan con una estructura bien definida, como el patrón Modelo-Vista-Controlador (MVC).

# RECOMENDACIONES

* En primer lugar, se sugiere utilizar un IDE estandarizado como Visual Studio .NET para aumentar la eficiencia en el desarrollo de proyectos. Esto ayudará a maximizar la productividad durante el proceso de programación.
* En segundo lugar, se aconseja mantener una documentación detallada del aplicativo en desarrollo. Esto beneficia tanto al lector, al proporcionar un mejor entendimiento del proyecto, como al desarrollador, al facilitar el aprendizaje y el seguimiento del progreso del desarrollo.

# REFERENCIAS

[1] Barrios, E. (2020, 20 de diciembre). Píldoras de C#: Task Parallel Library (TPL) Procesamiento Multihilo y asíncrono. DEV Community 👩‍💻👨‍💻. <https://dev.to/ebarrioscode/pildoras-de-c-task-parallel-library-tpl-procesamiento-multihilo-y-asincrono-18mp#:~:text=Un%20Thread%20(Hilo)%20es%20algo,es%20que%20los%20Threads%20comparten>

[2] "System.Threading Espacio de nombres". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/api/system.threading?view=net-7.0> (accedido el 18 de enero de 2023)

[3] "C #Ya - Delegados". Tutoriales Programacion Ya. <https://www.tutorialesprogramacionya.com/csharpya/detalleconcepto.php?codigo=217&amp;inicio=80> (accedido el 18 de enero de 2023).

[4] "ThreadStart Delegado (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstart?view=net-7.0> (accedido el 18 de enero de 2023).

[5] "ThreadPool Clase (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpool?view=net-7.0> (accedido el 18 de enero de 2023).

[6] "ThreadState EnumeraciÃ³n (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadstate?view=net-7.0> (accedido el 18 de enero de 2023).

[7] "ThreadPriority EnumeraciÃ³n (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadpriority?view=net-7.0> (accedido el 18 de enero de 2023).

[8] "ThreadLocal Clase (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/api/system.threading.threadlocal-1?view=net-7.0> (accedido el 18 de enero de 2023).

[9] "Subprocesos y subprocesamiento". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/standard/threading/threads-and-threading> (accedido el 18 de enero de 2023).

[10] "DestrucciÃ³n de subprocesos". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/standard/threading/destroying-threads> (accedido el 18 de enero de 2023).

[11] "Procesos e hilos Sistemas Operativos WebProgramacion Consultoría Informática". WebProgramacion Consultoría Informática. <https://webprogramacion.com/procesos-e-hilos/> (accedido el 16 de enero de 2023).

[12] "Programación con Hilos". prezi.com. <https://prezi.com/zligxav4or3z/programacion-con-hilos/> (accedido el 16 de enero de 2023).

[13] "HILOS EN C# - PARTE I". Blog | Construyendo el futuro de la web. <http://katawapps.blogspot.com/2015/12/hilos-en-c-parte-i.html> (accedido el 16 de enero de 2023).

[14] "Thread Class (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread?view=net-7.0> (accedido el 16 de enero de 2023).

[15] "Receta C# No. 4-7: Sincronizar la Ejecución de Múltiples Threads usando un Monitor". OrtizOL - Experiencias Construcción Software (xCSw). <https://ortizol.blogspot.com/2014/07/receta-csharp-no-4-7-sincronizar-la-ejecucion-de-multiples-threads-usando-un-monitor.html> (accedido el 16 de enero de 2023).

[16] "ThreadAbortException Class (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadabortexception?view=net-7.0> (accedido el 16 de enero de 2023).

[17] "ThreadExceptionEventArgs Class (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadexceptioneventargs?view=net-7.0> (accedido el 16 de enero de 2023).

[18] "ThreadExceptionEventHandler Delegate (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadexceptioneventhandler?view=net-7.0> (accedido el 16 de enero de 2023).

[19] "ThreadInterruptedException Class (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadinterruptedexception?view=net-7.0> (accedido el 16 de enero de 2023).

[20] "ThreadStartException Class (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadstartexception?view=net-7.0> (accedido el 16 de enero de 2023).

[21] "ThreadStateException Class (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.threadstateexception?view=net-7.0> (accedido el 16 de enero de 2023).

[22]"Pausa e interrupción de subprocesos". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/es-es/dotnet/standard/threading/pausing-and-resuming-threads> (accedido el 16 de enero de 2023).

[23] "Thread Constructor (System.Threading)". Microsoft Learn: Build skills that open doors in your career. <https://learn.microsoft.com/en-us/dotnet/api/system.threading.thread.-ctor?view=net-7.0> (accedido el 16 de enero de 2023).

[24] "Framework: ¿qué es y cuál es su función en Internet?" Rock Content - ES. <https://rockcontent.com/es/blog/framework/> (accedido el 27 de noviembre de 2022).

[25] H. Ordoñez. "Sistemas operativos, monotarea, multitarea, monousuario y multiusuario." Sistemas Operativos. http://heidiorba.blogspot.com/2016/05/sistemas-operativos-mono-tarea-multi.html (accedido el 27 de noviembre de 2022).

[26] M. Martínez. "Qué son los Patrones de Diseño de software / Design Patterns". Profile Software Services. https://profile.es/blog/patrones-de-diseno-de-software/ (accedido el 13 de noviembre de 2022).

[27] MDN Web Docs. "MVC - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN". MDN Web Docs. https://developer.mozilla.org/es/docs/Glossary/MVC#:~:text=MVC%20(Modelo-Vista-Controlador,de%20negocios%20y%20su%20visualización. (accedido el 14 de noviembre de 2022).

[28] Colaboradores de los proyectos Wikimedia. "Camel case - Wikipedia, la enciclopedia libre". Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/Camel_case> (accedido el 28 de diciembre de 2022).