

**Tema**

TALLER HILOS EN DOTNET

**AUTORES:**

Johao Morales

Maycol Tituaña

Alex Velasteguí

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS.MDU.CCNA.CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

07/08/2023

Contenido

[INTRODUCCIÓN 4](#_Toc124706015)

[OBJETIVO 4](#_Toc124706016)

[MARCO TEÓRICO 4](#_Toc124706017)

[C# 4](#_Toc124706018)

[.NET 4](#_Toc124706019)

[CLR (Common Language Runtime) 5](#_Toc124706020)

[HILOS 5](#_Toc124706021)

[ESTADOS 5](#_Toc124706022)

[INTERRUPCIONES 6](#_Toc124706023)

[SINCRONIZACIÓN 6](#_Toc124706024)

[Bloqueos 6](#_Toc124706025)

[Bloque condicional 6](#_Toc124706026)

[Desbloqueo 6](#_Toc124706027)

[MVC 6](#_Toc124706028)

[Modelo 7](#_Toc124706029)

[Vista 7](#_Toc124706030)

[Controlador 7](#_Toc124706031)

[FORM 7](#_Toc124706032)

[Controls 8](#_Toc124706033)

[Panel 8](#_Toc124706034)

[Button 8](#_Toc124706035)

[Método Show 10](#_Toc124706036)

[Método InitializeComponent 10](#_Toc124706037)

[EVENTARGS 11](#_Toc124706038)

[DELEGATE 11](#_Toc124706039)

[GRAPHICS 11](#_Toc124706040)

[Bitmap 11](#_Toc124706041)

[DESCRIPCIÓN TALLER 12](#_Toc124706042)

[DIBUJO DEL MOVIMIENTO DE UNA PELOTA 12](#_Toc124706043)

[DESARROLLO 12](#_Toc124706044)

[CONCLUSIONES 23](#_Toc124706045)

[RECOMENDACIONES 23](#_Toc124706046)

[Referencias 24](#_Toc124706047)

**ÍNDICE DE FIGURAS**

[Figura 1 4](#_Toc124706048)

[Figura 2 4](#_Toc124706049)

[Figura 3 6](#_Toc124706050)

[Figura 4 8](#_Toc124706051)

[Figura 5 8](#_Toc124706052)

[Figura 6 9](#_Toc124706053)

[Figura 7 9](#_Toc124706054)

[Figura 8 9](#_Toc124706055)

[Figura 9 10](#_Toc124706056)

[Figura 10 10](#_Toc124706057)

[Figura 11 11](#_Toc124706058)

[Figura 12 11](#_Toc124706059)

[Figura 13 12](#_Toc124706060)

[Figura 14 12](#_Toc124706061)

[Figura 15 13](#_Toc124706062)

[Figura 16 13](#_Toc124706063)

[Figura 17 13](#_Toc124706064)

[Figura 18 14](#_Toc124706065)

[Figura 19 14](#_Toc124706066)

[Figura 20 15](#_Toc124706067)

[Figura 21 15](#_Toc124706068)

[Figura 22 16](#_Toc124706069)

[Figura 23 16](#_Toc124706070)

[Figura 24 18](#_Toc124706071)

[Figura 25 19](#_Toc124706072)

[Figura 26 19](#_Toc124706073)

[Figura 27 19](#_Toc124706074)

[Figura 28 19](#_Toc124706075)

[Figura 29 20](#_Toc124706076)

[Figura 30 20](#_Toc124706077)

[Figura 31 22](#_Toc124706078)

[Figura 32 23](#_Toc124706079)

**ÍNDICE DE TABLAS**

[Tabla 1 17](#_Toc124706080)

[Tabla 2 20](#_Toc124706081)

[Tabla 3 22](#_Toc124706082)

# INTRODUCCIÓN

El lenguaje de programación c#, nos permite crear distintos tipos de aplicaciones, entre ellas, aplicaciones que permiten realizar trabajos de manera paralela mediante la implementación de hilos, estos hilos son ejecutados por el CLR (Common Language Runtime).

La implementación o utilización de hilos dentro de un software permite tener un mejor tiempo de respuesta, ya que se utilizan múltiples CPUs que permiten los hilos de un mismo proceso se ejecuten en diferentes CPUs y se ejecutan de manera simultánea.

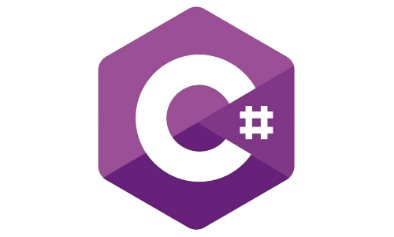
# OBJETIVO

Conocer y comprender, la funcionalidad y el desarrollo de hilos dentro de 3 aplicaciones multitareas utilizando el lenguaje de programación C#.

# MARCO TEÓRICO

## C#

Es una evolución que Microsoft realizó de este lenguaje, tomando lo mejor de los lenguajes C y C++, y ha continuado añadiéndole funcionalidades, tomando de otros lenguajes, como java, algo de su sintaxis evolucionada. Lo orientó a objetos para toda su plataforma NET, y con el tiempo adaptó las facilidades de la creación de código que tenía otro de sus lenguajes más populares, Visual Basic, haciéndolo tan polivalente y fácil de aprender como éste, sin perder ni un ápice de la potencia original de C [1].

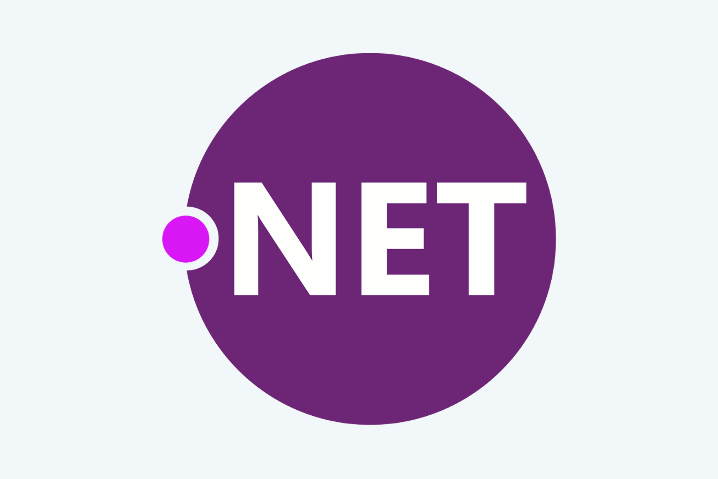


**Figura 1**

*Logotio de c#*

## .NET

.NET es una plataforma de código abierto para crear aplicaciones de escritorio, web y móviles que se pueden ejecutar de forma nativa en cualquier sistema operativo. El sistema .NET incluye herramientas, bibliotecas y lenguajes que admiten el desarrollo de software moderno, escalable y de alto rendimiento [2].



**Figura 2**

Símbolo representativo de .NET

## CLR (Common Language Runtime)

.NET proporciona un entorno en tiempo de ejecución denominado Common Language Runtime que ejecuta el código y proporciona servicios que facilitan el proceso de desarrollo.

Los compiladores y las herramientas exponen la funcionalidad de Common Language Runtime y permiten escribir código que se beneficia del entorno de ejecución administrado. El código que se desarrolla con un compilador de lenguaje y que tiene como destino el entorno de ejecución se denomina código administrado. El código administrado se beneficia de características como la integración entre lenguajes, el control de excepciones entre lenguajes, la seguridad mejorada, la compatibilidad con la implementación y el control de versiones, un modelo simplificado para interacción de componentes, y servicios de generación de perfiles y depuración [3].

## HILOS

Un hilo es una línea de ejecución de un proceso. Todo proceso parte inicialmente con un único hilo principal, aunque el sistema operativo ofrece llamadas al sistema que permiten al programador crear y destruir hilos. Por tanto, un proceso está compuesto por uno o más hilos.

Para crear un hilo dentro de C#, debemos crear un objeto del tipo Thread y en el constructor recibe una función como parámetro, esta función será la que ejecutan cuando se llame a la función Star(), que corresponde a cuando el hilo comienza su ejecución.

### ESTADOS

Los hilos pueden encontrarse con los siguientes estados [4]:

1. **Nuevo:** El hilo ha sido creado, pero aún no ha sido activado. Cuando se active, pasará al estado preparado.
2. **Preparado:** El hilo está activo y está a la espera de que le sea asignada la UCP
3. **En ejecución:** El hilo está activo y le ha sido asignada la UCP (sólo los hilos activos, en estado preparado, pueden ser ejecutados)
4. **Bloqueado:** El hilo espera que otro elimine el bloqueo. Un hilo bloqueado puede estar: dormido (bloqueado durante una cantidad de tiempo determinada, después de la cual despertará y pasará al estado preparado), o esperando (el hilo está esperando a que ocurra alguna cosa)
5. **Muerto:** El hilo ha finalizado (está muerto) pero todavía no ha sido recogido por su padre.

## INTERRUPCIONES

Los objetos de clase Thread cuentan con un método .interrupt() que permite al hilo ser interrumpido. En realidad, la interrupción simplemente cambia un flag del hilo para marcar que ha de ser interrumpido, pero cada hilo debe estar programado para soportar su propia interrupción.

Si el hilo invoca un método que lance la excepción InterruptedException, tal como el método sleep(), entonces en este punto del código terminaría la ejecución del método run() del hilo [5].

## SINCRONIZACIÓN

Cuando uno o más hilos necesitan acceso a un recurso compartido que solo puede ser usado por un hilo a la vez, es necesario coordinar las actividades de estos hilos, para que cada hilo pueda efectuar su trabajo sin necesidad de que se vea interrumpido por otro hilo.

### Bloqueos

Las funciones indicadas anteriormente también se pueden complementar con funciones como Lock(), la cual garantiza que un nuevo hilo no ingrese a una sección crítica sin que el anterior hilo haya terminado de ejecutarse [6].

### Bloque condicional

Puede ser necesario esperar a que un determinado hilo haya finalizado su tarea para continuar, esto se puede realizar llamando al método Join() de dicho hilo, que bloqueará el siguiente hilo hasta que el hilo anterior haya finalizado [6].

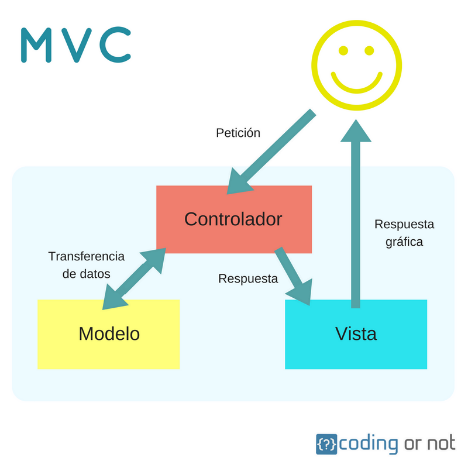
### Desbloqueo

En el caso de Lock() bastará con esperar a que el hilo anterior haya terminado de ejecutarse, ya que esta función se encarga de asegurarse ese comportamiento mismo [6].

## MVC

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo [7].



**Figura 3**

Representación gráfica del patrón MVC

### Modelo

Es la capa donde se trabaja con los datos, esta capa está apegada a la base de datos, es decir, es la representación de las tablas, por lo tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado.

Como se mencionó anteriormente, los datos estarán contenidos en una base de datos, por lo que en los modelos tendremos las funciones necesarias para acceder a las tablas y harán las correspondientes consultas [7].

### Vista

Contienen el código de las interfaces gráficas que contiene la aplicación, es decir, el código que permitirá renderizar los estados de nuestra aplicación en HTML o PHP, pues nos permiten mostrar la salida.

En la vista generalmente se trabaja con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generará la salida, tal como nuestra aplicación requiera [7].

### Controlador

El controlador contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

Es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo [7].

## FORM

Es una clase que permite la representación de una ventana o un cuadro de diálogo que constituye la interfaz de usuario de una aplicación en C#.

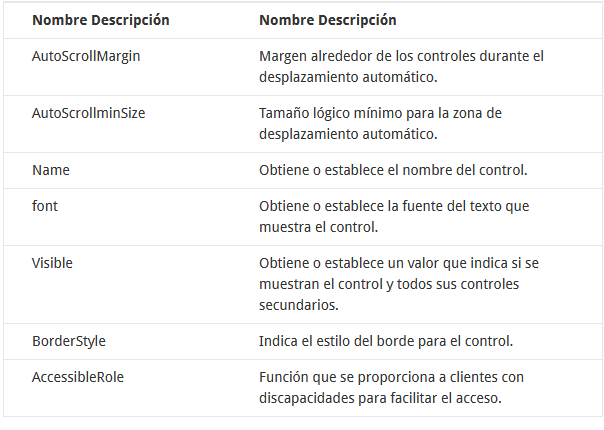
### Controls

También conocido como controles de usuario, agrupa de manera lógica los controles para obtener una entidad reutilizable.

Se pueden construir con el diseñador de pantallas igual que con los formularios, solo basta arrastrar los controles desde la caja de herramientas al diseñador de pantallas, a continuación, vamos a presentar y resumir algunos de los tipos de controladores que suelen ser muy usados [8].

### Panel

El panel es un control contenedor para albergar a un grupo de controles secundarios similares. Uno de los principales usos que se ha encontrado para un panel de control es cuando se necesita mostrar y ocultar un grupo de controles. En lugar de mostrar y ocultar controles individuales, simplemente puede ocultar y mostrar un solo panel y todos los controles secundarios [9].



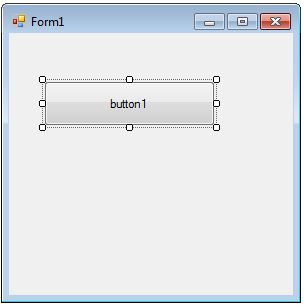
**Figura 4**

Propiedades de Panel

### Button

El control [Button](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.button) de Windows Forms permite al usuario hacer clic en él para llevar a cabo una acción. Al hacer clic en el botón, parece como si se hubiera presionado y soltado. Cada vez que el usuario hace clic en un botón, se invoca el controlador de eventos [Click](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.control.click). Debe colocar el código en el controlador de eventos [Click](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.control.click) para realizar la acción que elija.

El control [Button](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.button) también puede mostrar imágenes mediante las propiedades [Image](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.buttonbase.image) y [ImageList](https://learn.microsoft.com/es-es/dotnet/api/system.windows.forms.buttonbase.imagelist). Para obtener más información, vea [Procedimiento para establecer la imagen mostrada por un control de Windows Forms](https://learn.microsoft.com/es-es/dotnet/desktop/winforms/controls/how-to-set-the-image-displayed-by-a-windows-forms-control?view=netframeworkdesktop-4.8).

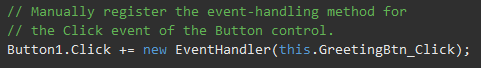


**Figura 5**

Botón en C#

#### Método Click

Es una función que se ejecuta o se lanza al momento que se hace click en un botón, en la Figura 6 se presenta como agregar este método en un botón.

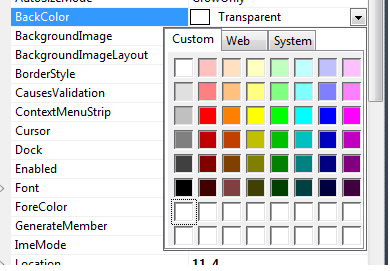


**Figura 6**

Adición de método click

#### BackColor

Esta propiedad se encarga de establecer o de obtener el color de fondo de un control.

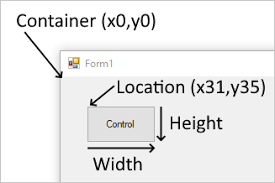


**Figura 7**

Colores disponibles para la propiedad BackColor

#### Location

Esta propiedad permite obtener o establecer las coordenadas de la esquina superior izquierda del control en relación con la esquina superior izquierda de su contenedor.



**Figura 8**

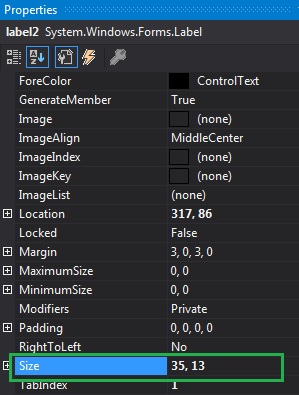
Ejemplo de la propiedad Location

#### Name

Esta propiedad permite obtener o establecer el nombre de un control.

#### Size

Esta propiedad se encarga del tamaño de un control, especifica la altura y el ancho.

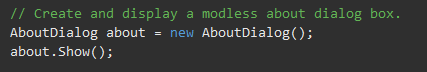


**Figura 9**

Propiedad Size

### Método Show

Este método permite mostrar un control al usuario.



**Figura 10**

Ejemplo de código del método show

### Método InitializeComponent

Este método se encargar de cargar lá página compilada de un componente.

## EVENTARGS

Representa la clase base para las clases que contienen datos de eventos, y proporciona un valor que se utilizará para eventos que no incluyen datos de eventos.

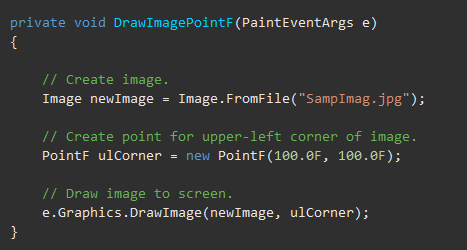
## DELEGATE

Un delegate en C# es un tipo de dato como lo es una estructura, o una clase, estos tipos definen propiedades, métodos y algunas veces eventos, por su parte los delegates se encargan de referenciar métodos que coincidan con una firma compatible determinada por los parámetros de entrada y tipo de retorno.

## GRAPHICS

Encapsula una superficie de dibujo de GDI+. Esta clase no puede heredarse. Pertenece al espacio System.Drawing.

En la Figura 11 se muestra un ejemplo de aplicación.

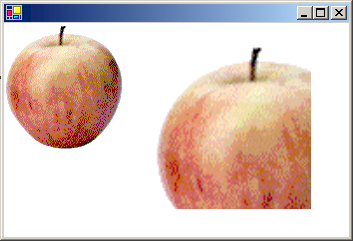


**Figura 11**

Ejemplo de aplicación de Graphics

### Bitmap

Encapsula un mapa de bits de GDI+, que está formado por los datos de píxeles de una imagen de gráficos y sus atributos. [Bitmap](https://learn.microsoft.com/es-es/dotnet/api/system.drawing.bitmap?view=dotnet-plat-ext-7.0) es un objeto que se utiliza para trabajar con imágenes definidas mediante datos de píxeles.



**Figura 12**

Ejemplo de bitmap

# DESCRIPCIÓN TALLER

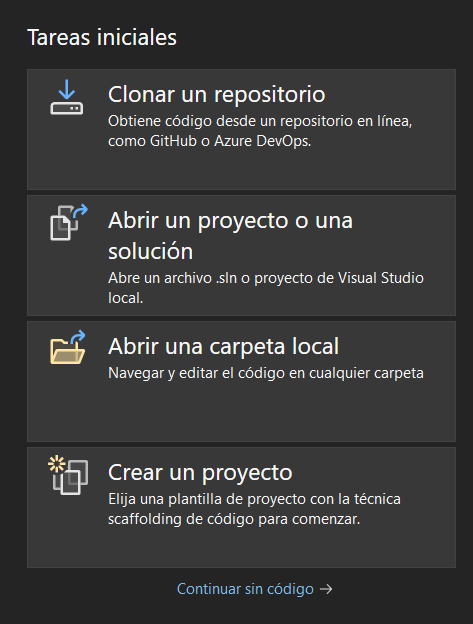
El presente taller consiste en realizar una simulación de transferencias de un banco, donde se aplican los conceptos de sincronización y de bloqueo de hilos, es proyecto se pudo realizar en el lenguaje de programación C#.

## DIBUJO DEL MOVIMIENTO DE UNA PELOTA

### DESARROLLO

#### CREACIÓN DEL PROYECTO Y ESTRUCTURA

Para crear un proyecto debemos hacer click en crear nuevo proyecto.



**Figura 13**

Creación de nuevo proyecto

Y escogemos una aplicación de Consola.

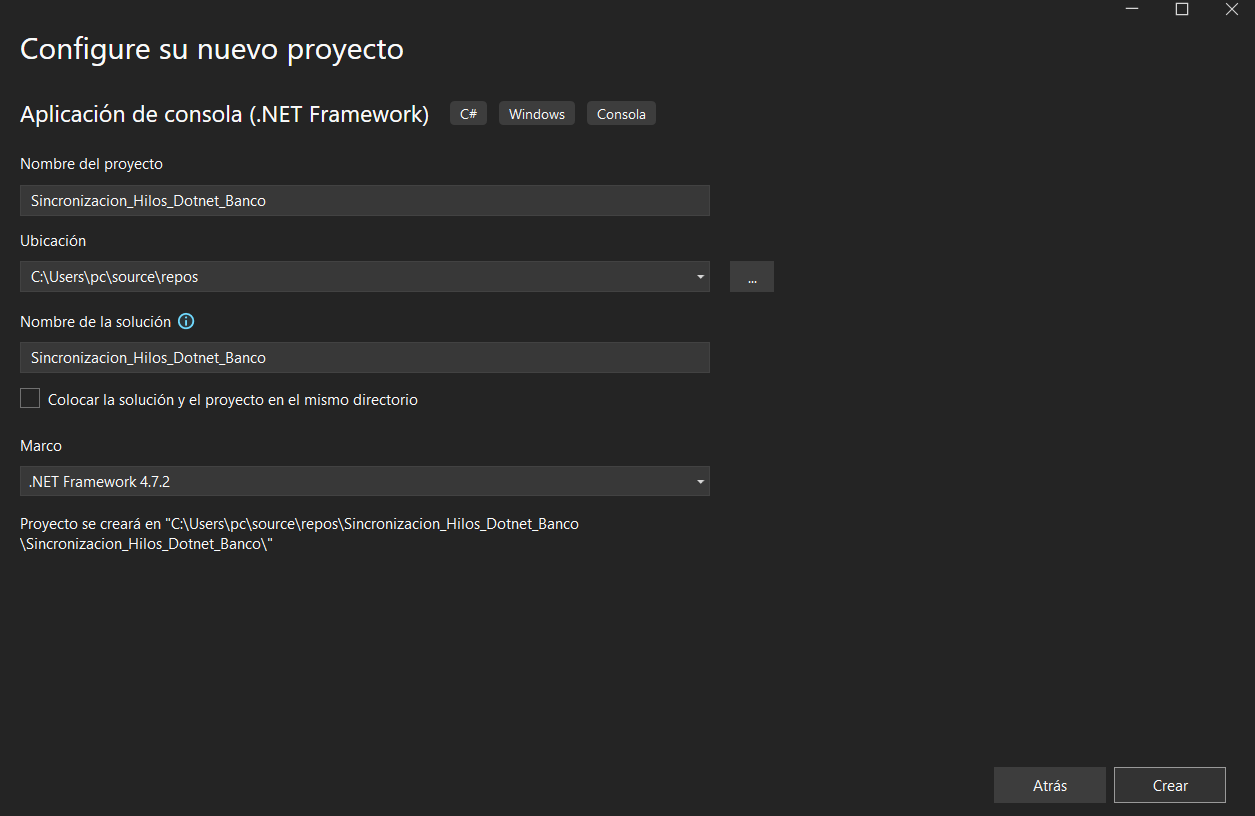
Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

**Figura 14**

Selección del tipo de proyecto

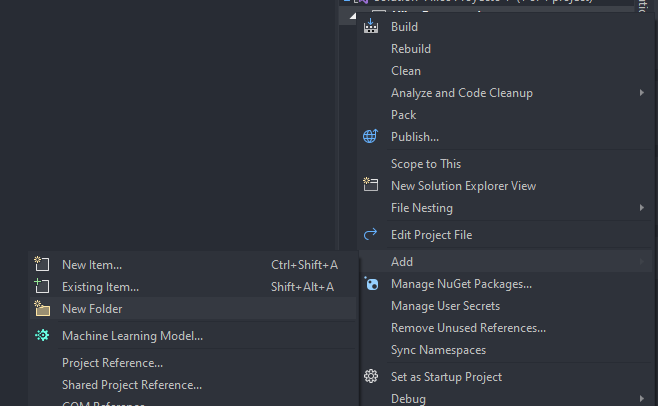
Damos un nombre al proyecto en este caso “Sincronizacion\_Hilos\_Dotnet\_Banco”



**Figura 14**

Nombre del proyecto

Para estructurar nuestro proyecto aplicando el patrón MVC, debemos crear carpetas, para ello hacemos click derecho en la solución y en la opción Add seleccionamos new Folder o nueva carpeta.



**Figura 15**

Creación de nueva carpeta

Y asignamos el nombre de la carpeta.

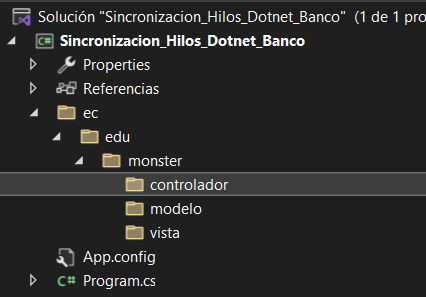
Texto

Descripción generada automáticamente

**Figura 16**

Asignación del nombre de la carpeta

La creación de las carpetas la haremos cuantas veces sea necesaria para poder tener la estructura que usamos en los proyectos en JAVA, a continuación, se muestra el directorio final.



**Figura 17**

Directorio del proyecto

#### VISTA

Para este proyecto, no se desarrolló ninguna vista, ya que es una aplicación de consola.

#### MODELO

**Creación del modelo Banco**

Para crear el modelo, tendremos que crear una nueva clase, para ello, hacemos click derecho en la carpeta de modelo, y apoyando el mouse sobre la opción Add, hacemos click en Class.

Captura de pantalla de un celular

Descripción generada automáticamente

**Figura 18**

Creación de nueva clase

Y al modelo, le asignamos el nombre Banco, y le damos click en Add.

Captura de pantalla de computadora

Descripción generada automáticamente

**Figura 19**

Asignación del nombre del modelo banco

En el constructor del modelo Banco, tenemos la creación de las 100 cuentas, iniciamos las cuentas con un monto inicial de 200.

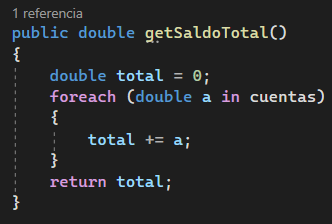
Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

**Figura 20**

Inicialización de 100 cuentas en constructor

Para poder obtener el monto total de todas las 100 cuentas creadas en el constructor, realizamos la función getSaldoTotal, que recorre las cuentas y realiza la suma de cada una de estas.



**Figura 21**

Creación de función getSaldoTotal

En el método transferencia, se realiza una simulación, desde una cuenta origen a una cuenta destino, se descuenta un monto recibido por parámetro y se muestra por pantalla el monto retirado y el saldo total de dicha cuenta, es importante tener en cuenta que, estas cuentas son tomadas del arreglo creado en el constructor del modelo.

Captura de pantalla de un celular

Descripción generada automáticamente

**Figura 22**

Creación de función transferencia

Finalmente, tendríamos el código del modelo, de la siguiente manera.

Texto

Descripción generada automáticamente

**Figura 23**

Código del modelo Banco

**Tabla 1**

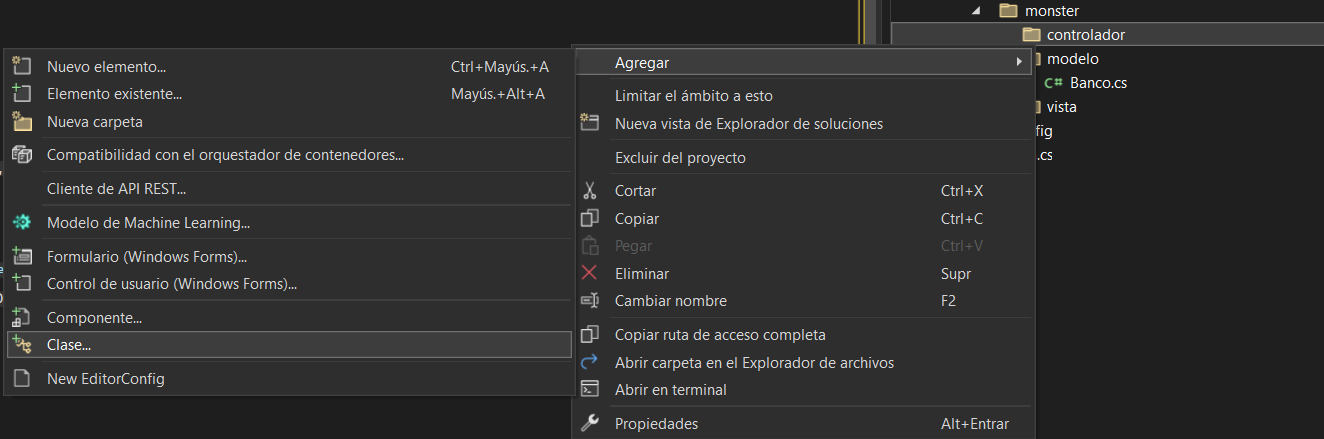
Código del modelo Banco

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Sincronizacion\_Hilos\_Dotnet\_Banco.ec.edu.monster.modelo  {  class Banco  {  private double[] cuentas;  //Constructor inicializa 100 cuentas con saldo de 2000 dolares  public Banco()  {  this.cuentas = new double[100];  for (int i = 0; i < cuentas.Length; i++)  {  cuentas[i] = 2000;  }  }  public void transferencia(int cuentaOrigen, int cuentaDestino, double cantidad)  {  while (cuentas[cuentaOrigen] < cantidad)  {  return;  }  cuentas[cuentaOrigen] -= cantidad;  Console.WriteLine($"Transferir ${cantidad} de la cuenta {cuentaOrigen} para la cuenta {cuentaDestino}. \r\n");  cuentas[cuentaDestino] += cantidad;  Console.WriteLine($"Saldo total : {cuentas[cuentaDestino]}.\r\n");  }  //Se obtiene el saldo total de la cuenta  public double getSaldoTotal()  {  double total = 0;  foreach (double a in cuentas)  {  total += a;  }  return total;  }  }  } |

#### CONTROLADOR

**Creación del controlador EjecucionTransferencias**

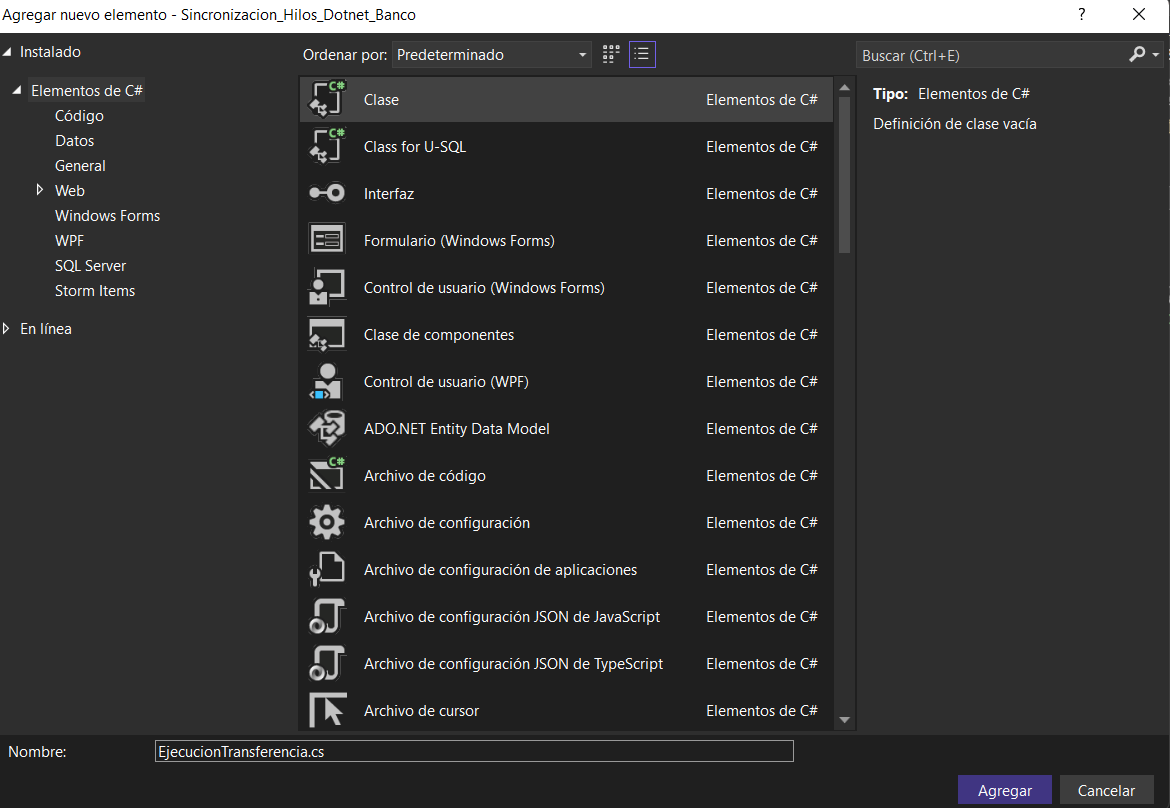
Para crear el controlador, tendremos que crear una nueva clase, para ello, hacemos click derecho en la carpeta de controlador, y apoyando el mouse sobre la opción Add, hacemos click en Class.



**Figura 24**

Creación de nueva clase

Y al modelo, le asignamos el nombre EjecucionTransferencias, y le damos click en Add.



**Figura 25**

Asignación del nombre del controlador

Los atributos que tendrá este controlador, será, un objeto de tipo Banco, un monto máximo de transferencia, la cuenta de origen y el hilo con el que el controlador va a trabajar.

Texto

Descripción generada automáticamente

**Figura 26**

Atributos del controlador

La función comenzar será la encargada de ejecutarse cuando el hilo que pertenece a esta clase sea ejecutado, en esta función creamos valores aleatorios que servirán de simulación de una transferencia de una cuenta de banco de origen a una de destino.

Texto

Descripción generada automáticamente

**Figura 27**

Función comenzar

En el constructor del controlador, vamos a instanciar, tanto las variable de deLaCuenta, cantidadMax y banco, pero también vamos a instanciar el hilo con la función comenzar, para que de esta manera, cuando el hilo ejecute el método Start, esta función se ejecute.

Texto

Descripción generada automáticamente

**Figura 28**

Constructor del controlador

Como última función del controlador, tenemos la función getHilo, que se encarga de retornar nuestro hilo para que sea ejecutado desde el archivo Program.cs.



**Figura 29**

Función getHilo

De esta manera, tenemos el código del controlador.

Texto

Descripción generada automáticamente

**Figura 30**

Código del controlador

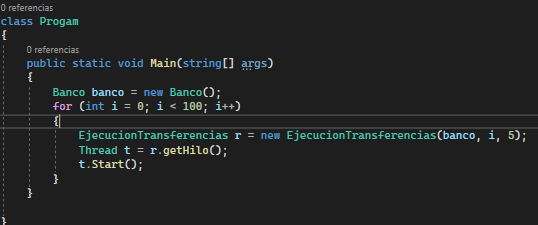
**Tabla 2**

Código del controlador

|  |
| --- |
| using Sincronizacion\_Hilos\_Dotnet\_Banco.ec.edu.monster.modelo;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  namespace Sincronizacion\_Hilos\_Dotnet\_Banco.ec.edu.monster.controlador  {  class EjecucionTransferencias  {  private Banco banco;  private int deLaCuenta;  private int cantidadMax;  private Thread t;  public EjecucionTransferencias(Banco banco, int deLaCuenta, int cantidadMax)  {  this.banco = banco;  this.deLaCuenta = deLaCuenta;  this.cantidadMax = cantidadMax;  t = new Thread(comenzar);  }  public Thread getHilo() { return t; }  public void comenzar()  {  int limite = cantidadMax + 1;  var r = new Random();  double cantidad = r.NextDouble() \* (10 - (limite)) + (limite);  int cantidadEntera = (int)Math.Round(cantidad);  cantidad = cantidadEntera;  deLaCuenta = r.Next(1, 11);  int paraLaCuenta = r.Next(1, 11);  while (true)  {  banco.transferencia(deLaCuenta, paraLaCuenta, cantidad);  deLaCuenta = r.Next(1, 11);  paraLaCuenta = r.Next(1, 11);  cantidad = r.NextDouble() \* (10 - (limite)) + (limite);  Thread.Sleep(500);  }  }  }  } |

#### Archivo Principal Program.cs

Este archivo es importante para el proyecto, ya que desde aquí se ejecuta nuestro proyecto, es por eso que cuando usamos el patrón MVC, tenemos que invocar a los controladores en este archivo, por eso instanciamos nuestro controlador ya creado, junto con nuestro objeto modelo banco, y la cuenta de origen, que consiste en el iterador del ciclo for, para que esto sea un proceso sin final y se irá retirando de 5 en 5, también se pude apreciar, como obtenemos el hilo y lo ejecutamos con el método Start.



**Figura 31**

Código del archivo Program.cs

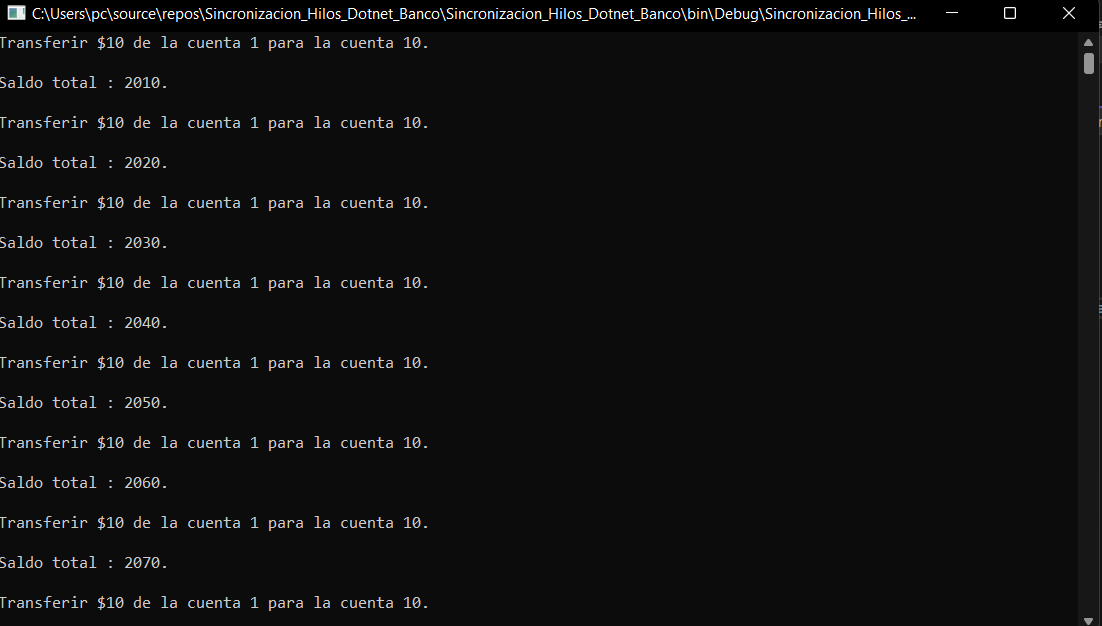
**Tabla 3**

Código del archivo Program.cs

|  |
| --- |
| using Sincronizacion\_Hilos\_Dotnet\_Banco.ec.edu.monster.controlador;  using Sincronizacion\_Hilos\_Dotnet\_Banco.ec.edu.monster.modelo;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading;  using System.Threading.Tasks;  namespace Sincronizacion\_Hilos\_Dotnet\_Banco  {  class Progam  {  public static void Main(string[] args)  {  //Se crea una instancia del modelo banco.  //El constructor de la clase crea 100 cuentas con 2000 dolares.  Banco banco = new Banco();  //El for simula las transacciones a las cuentas creadas.  for (int i = 0; i < 1000; i++)  {  EjecucionTransferencias r = new EjecucionTransferencias(banco, i, 5);  //Se crea una instancia de un hilo y se ejecuta.  Thread t = r.getHilo();  t.Start();  }  }  }  } |

#### EJECUCIÓN

Como se puede apreciar en la Figura 32, ya se realizan las transferencias y se muestra el saldo total, por lo que el proyecto ha sido realizado exitosamente.



**Figura 32**

Ejecución del proyecto 3

# CONCLUSIONES

* Se ha evidenciado de manera práctica como el lenguaje C# brinda soporte para la creación y administración de hilos. De igual manera se ha entendido el funcionamiento de los hilos y sus estados.
* Se ha demostrado la importancia de los bloques sincronizados y los bloqueos para conseguir una sincronización efectiva en la ejecución de multihilos y de esta manera evitar problemas que puedan surgir en aplicaciones multitarea.
* Se ha logrado identificar que el manejo de hilos tanto en los lenguajes de programación JAVA y C#.

# RECOMENDACIONES

* Se recomienda asignar nombres identificativos para cada componente gráfico, ya que a la hora de agregar alguna función, ya sea un evento, resultará más fácil su manejo.
* Se recomienda revisar las librerías tanto gráficas como para el manejo de hilos, ya que en el lenguaje de programación JAVA, las librerías cuentan con distintas funciones.

# Referencias

|  |  |
| --- | --- |
| [1] | BeSoftware, «BSW,» 15 10 2022. [En línea]. Available: https://bsw.es/que-es-c/. |
| [2] | AWS Amazon, «Amazon AWS,» s.f ¿Qué es .NET?. [En línea]. Available: https://aws.amazon.com/es/what-is/net/. [Último acceso: 9 1 2023]. |
| [3] | M. Wenzel, «Introducción a Common Language Runtime,» 28 11 2022. [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/standard/clr. [Último acceso: 9 1 2023]. |
| [4] | G. Elordury, «Estados de un hilo,» 7 5 2017. [En línea]. Available: https://javaparajavatos.wordpress.com/2017/05/07/estados-de-un-hilo/. [Último acceso: 9 1 2023]. |
| [5] | Dpto. Ciencias de computación e IA, «Hilos,» 17 10 2012. [En línea]. Available: http://www.jtech.ua.es/dadm/restringido/java/sesion05-apuntes.html#:~:text=Interrupci%C3%B3n%20de%20un%20hilo,-Los%20objetos%20de&text=interrupt()%20que%20permite%20al,para%20soportar%20su%20propia%20interrupci%C3%B3n.. [Último acceso: 9 1 2023]. |
| [6] | A. Kumar, «Monitor And Lock,» 29 5 2019. [En línea]. Available: https://www.c-sharpcorner.com/uploadfile/de41d6/monitor-and-lock-in-c-sharp/. [Último acceso: 9 1 2023]. |
| [7] | M. Álvarez, «Qué es MVC,» 28 7 2020. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html. [Último acceso: 11 1 2023]. |
| [8] | H. Jerome, «Programacion En C#,» 2017. [En línea]. Available: https://www.ediciones-eni.com/open/mediabook.aspx?idR=0b128b0b0047e168fb84560d5b827a13. [Último acceso: 11 1 2023]. |
| [9] | luna, «Programacion En C#,» 9 3 2014. [En línea]. Available: https://csharmaniacos.wordpress.com/2014/03/09/panel/. [Último acceso: 11 1 2023]. |