**Channel – corner**

Channel یک افزونه ای اساسی اضافه شده به برنامه نویسی Asynchronus است که یک مسیری ارائه می دهد برای انتقال داده بین تولید کننده و مصرف کننده در به یک شیوه ای thread-safe

Channel سبب افزایش کارایی و مقیاس پذیری در برنامه کاربردی می شود. یک وسیله ارتباطی انعطاف پذیر و کارآمد است.

using System.Threading.Channels;

Console.WriteLine("Channels In .NET");

// Create an unbounded channel

var channel = Channel.CreateUnbounded<int>();

/ Producer writing data to the channel

async Task ProduceAsync()

{

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

{

await channel.Writer.WriteAsync(i);

Console.WriteLine($"Produced: {i}");

}

channel.Writer.Complete();

}

// Consumer reading data from the channel

async Task ConsumeAsync()

{

while (await channel.Reader.WaitToReadAsync())

{

while (channel.Reader.TryRead(out var item))

{

Console.WriteLine($"Consumed: {item}");

}

}

}

// Run producer and consumer asynchronously

var producerTask = ProduceAsync();

var consumerTask = ConsumeAsync();

// Wait for both tasks to complete

await Task.WhenAll(producerTask, consumerTask);

Console.ReadLine();

Bounded and Unbounded Channels

می توان بین محدود بدون اندازه channel و unbounded بدون یا نامحدود بدون انتخاب کرد.

Completion

علامت گذاری این که channel کامل شده است و جریان داده به اتمام رسیده.

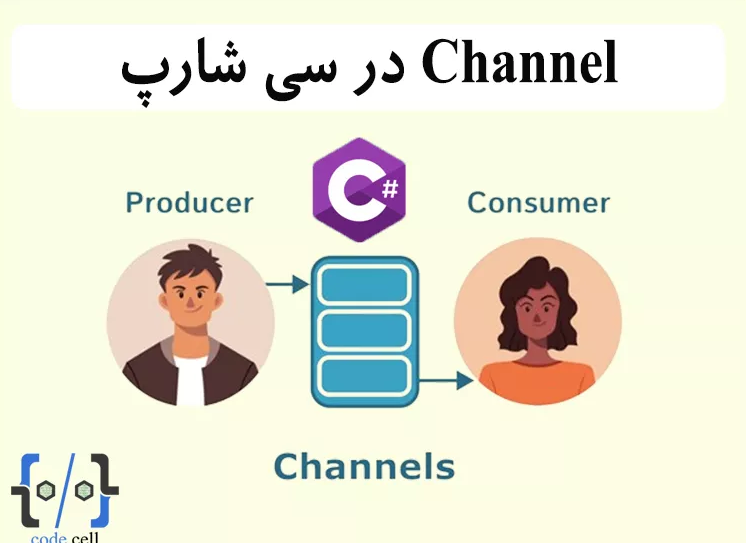
Cancellation

استفاده از cancellation token برای مدیریت چرخه عمر کانال.

Multiple Readers/Writers:

امکان استفاده آسان توسط چندین خواننده و نویسنده.

کاربرد مهم کانال ها در handle کردن asynchronous data streaming است. و امکان ارتباط موثر بین بخش های مختلف یک برنامه را امکان پذیر می کند به همراه عملکرد بالا و thread safety



**در سی شارپ چیست؟**

Channel در سی شارپ یک ویژگی هستند که در .NET Framework برای تسهیل ارتباط بین بخش های مختلف یک برنامه کاربردی معرفی شده اند. Channel ها مکانیزمی هستند که به thread ها یا task ها اجازه می دهد تا داده ها را به صورت ایمن و async بدون مسدود کردن یکدیگر یا ایجاد race condition مبادله کنند.

Channel .های سی شارپ برای سناریوهایی ایده‌آل هستند که در آن باید داده‌ها را بین تولیدکننده یا Producer نویسنده یا writer و مصرف‌کننده یا Consumer خواننده یا reader بدون نگرانی در مورد synchronization یا همگام‌سازی thread ها توزیع کنید.

## دلایل استفاده از Channel های سی شارپ:

**ارتباط ناهمزمان یا Asynchronous :** کانال های سی شارپ ارتباط ناهمزمان بین thread یا task هارا فعال می کنند. تولیدکنندگان یا Producer ها می توانند داده ها را بدون اینکه منتظر بمانند تا consumer یا مصرف کننده آن را پردازش کند، در کانال بنویسند و مصرف کننده می تواند داده ها را از کانال به صورت ناهمزمان بخواند.

**Thread safety:** کانال ها به طور ذاتی hread safety را فراهم می کنند. آن ها دسترسی به حافظه مشترک (کانال) را synchronize یا همگام می کنند، که این موضوع باعث می شود از race condition و سایر مشکلاتی که می تواند در برنامه های multithreaded رخ دهد جلوگیری شود.

**جداسازی تولید کننده و مصرف کننده:** تولید کننده و مصرف کننده نیازی به دانستن یکدیگر ندارند. این جداسازی قابلیت نگهداری را تسهیل می‌کند و اجازه می‌دهد تا اجزاء به طور مستقل توسعه و تست شوند.

**بافر کردن:** با کانال های C#، می توانید ظرفیت بافر را مشخص کنید. این به تولید کننده اجازه می دهد تا داده ها را حتی اگر مصرف کننده بلافاصله آماده پردازش آن نباشد، در کانال بنویسد.

**Cancellation condition:** می توانید Cancellation condition را برای توقف پردازش در صورت رعایت شرایط خاص تعریف کنید.

**کنترل جریان داده:** کانال ها را می توان در سناریوهایی استفاده کرد که در آن شما نیاز به کنترل جریان داده بین بخش های مختلف برنامه خود دارید. این به ویژه زمانی مفید است که task را به صورت parallel اجرا می کنید و می خواهید اطمینان حاصل کنید که منابع خاصی بیش از حد بارگذاری نمی شوند.

### Channel چیست؟

Channel<T> در فضای نام System.Threading.Channels (از .NET Core 3 به بعد) معرفی شده.  
کارش شبیه یک **صف thread-safe** است، برای سناریوهایی که می‌خواهی داده‌ها بین چند **تولیدکننده (producer)** و **مصرف‌کننده (consumer)** رد و بدل شوند.

این یعنی می‌توانی یک جریان داده بسازی که یک یا چند تا thread یا task داخلش داده بریزن و یک یا چند تا task دیگه اون داده‌ها رو بخونن، بدون این‌که نگران lock و مدیریت همزمانی باشی.

### کاربردهای مهم Channel

* **پردازش غیرهمزمان (async)**: مثلاً داده‌ها رو از یک منبع بیرونی می‌گیری و می‌خوای پردازش کنی.
* **مدل تولیدکننده/مصرف‌کننده**: خیلی ساده این الگو رو پیاده می‌کنه.
* **جایگزین BlockingCollection**: قبلاً بیشتر از BlockingCollection<T> استفاده می‌کردیم، اما Channel سبک‌تر و modern-تره.
* **کنترل فشار (backpressure)**: می‌تونی کانال رو bounded تعریف کنی تا اگه مصرف‌کننده عقب افتاد، تولیدکننده نتونه بی‌نهایت داده بریزه.

var channel = Channel.CreateUnbounded<int>();

var producer = Task.Run(async () =>

{

for (int i = 1; i <= 5; i++)

{

await channel.Writer.WriteAsync(i);

Console.WriteLine($"Produced: {i}");

await Task.Delay(500);

}

channel.Writer.Complete(); // یعنی دیگه چیزی نوشته نمی‌شه

});

// Consumer

var consumer = Task.Run(async () =>

{

await foreach (var item in channel.Reader.ReadAllAsync())

{

Console.WriteLine($"Consumed: {item}");

await Task.Delay(1000);

}

});

await Task.WhenAll(producer, consumer);

Produced: 1

Consumed: 1

Produced: 2

Produced: 3

Consumed: 2

Produced: 4

Produced: 5

Consumed: 3

Consumed: 4

Consumed: 5