C#

Recursos gerenciados: Ao falar de recursos gerenciados estamos falando na prática de memória gerenciada. A memoria gerenciada é aquela associada a objetos na heap que, quando não tem mais nenhuma referência no código, são coletados, eventualmente, pelo Garbage Collector. Também estamos falando dos valores armazenados na stack que são desalocados quando há um descarte de um stack frame.

Recursos não gerenciados: Estão entre os recursos não gerenciados a memória alocada que não é controlada pelo garbage Collector, como aquela alocada em rotinas escritas em C++. Como exemplo de recursos não gerenciados os **handles de arquivo, sockets** e **outros recursos de rede, conexões com banco de dados, objetos GDI.**

Handle: É um identificador gerado pelo SO atribuído a cada processo em funcionamento. Por exemplo, quando o usuário abre um arquivo, ele vê apenas o nome do arquivo, porém o SO identifica o arquivo pelo handle atribuído a ele. Cada janela aberta no seu SO tem um handle.

Método Dispose: objetos que utilizam recursos não gerenciados geralmente precisam elaborar alguma estratégia de desalocação explicita, geralmente através do método dispose. Objetos não gerenciados não são desalocados automaticamente em .NET. Se um objeto implementa o método Dispose, então, este deve ser evocado explicitamente no código (ou implicitamente, através da utililazação da instrução using).

Stream Classe

Fornece uma exibição de uma sequência de bytes. Esta é uma classe abstrata.

[Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) é a classe base abstrata de todos os fluxos. *Um fluxo é uma abstração de uma sequência de bytes*, como um arquivo, um dispositivo de arquivos entrada/saída, um pipe de comunicação de inter- processo, ou um soquete TCP/IP. A [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) classe e suas classes derivadas fornecem uma exibição genérica desses diferentes tipos de entrada e saída e isolam o programador dos detalhes específicos do sistema operacional e dos dispositivos subjacentes.

Fluxos envolvem estas três operações fundamentais:

* Você pode ler de fluxos. A leitura é a transferência de dados de um fluxo para uma estrutura de dados, como uma matriz de bytes.
* Você pode gravar em fluxos. A gravação é a transferência de dados de uma estrutura de dados para um fluxo.
* Os fluxos podem dar suporte à busca. A busca refere-se à consulta e à modificação da posição atual em um fluxo. A funcionalidade de busca depende do tipo de armazenamento de backup que um fluxo tem. Por exemplo, os fluxos de rede não têm um conceito unificado de uma posição atual e, portanto, normalmente não dão suporte à busca.

Alguns dos fluxos mais comumente usados que herdam de [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) são [FileStream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.filestream?view=net-5.0) , e [MemoryStream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.memorystream?view=net-5.0) .

Dependendo da fonte de dados ou do repositório subjacente, os fluxos podem dar suporte a apenas alguns desses recursos. Você pode consultar um fluxo para seus recursos usando as [CanRead](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.canread?view=net-5.0) Propriedades, [CanWrite](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.canwrite?view=net-5.0) e [CanSeek](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.canseek?view=net-5.0) da [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) classe.

Os [Read](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.read?view=net-5.0) [Write](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.write?view=net-5.0) métodos e lêem e gravam dados em uma variedade de formatos. Para fluxos que dão suporte à busca, use os [Seek](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.seek?view=net-5.0) [SetLength](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.setlength?view=net-5.0) métodos e e as [Position](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.position?view=net-5.0) [Length](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.length?view=net-5.0) Propriedades e para consultar e modificar a posição e o comprimento atuais de um fluxo.

Esse tipo implementa a interface [IDisposable](https://docs.microsoft.com/pt-br/dotnet/api/system.idisposable?view=net-5.0). Quando você terminar de usar o tipo, deverá descartá-lo direta ou indiretamente. Para descartar o tipo diretamente, chame o método [Dispose](https://docs.microsoft.com/pt-br/dotnet/api/system.idisposable.dispose?view=net-5.0) dele em um bloco try/catch. Para descartá-lo indiretamente, use um constructo de linguagem como using ( em C#) ou Using (em Visual Basic). Saiba mais na seção "Como usar um objeto que implementa IDisposable" no tópico da interface [IDisposable](https://docs.microsoft.com/pt-br/dotnet/api/system.idisposable?view=net-5.0).

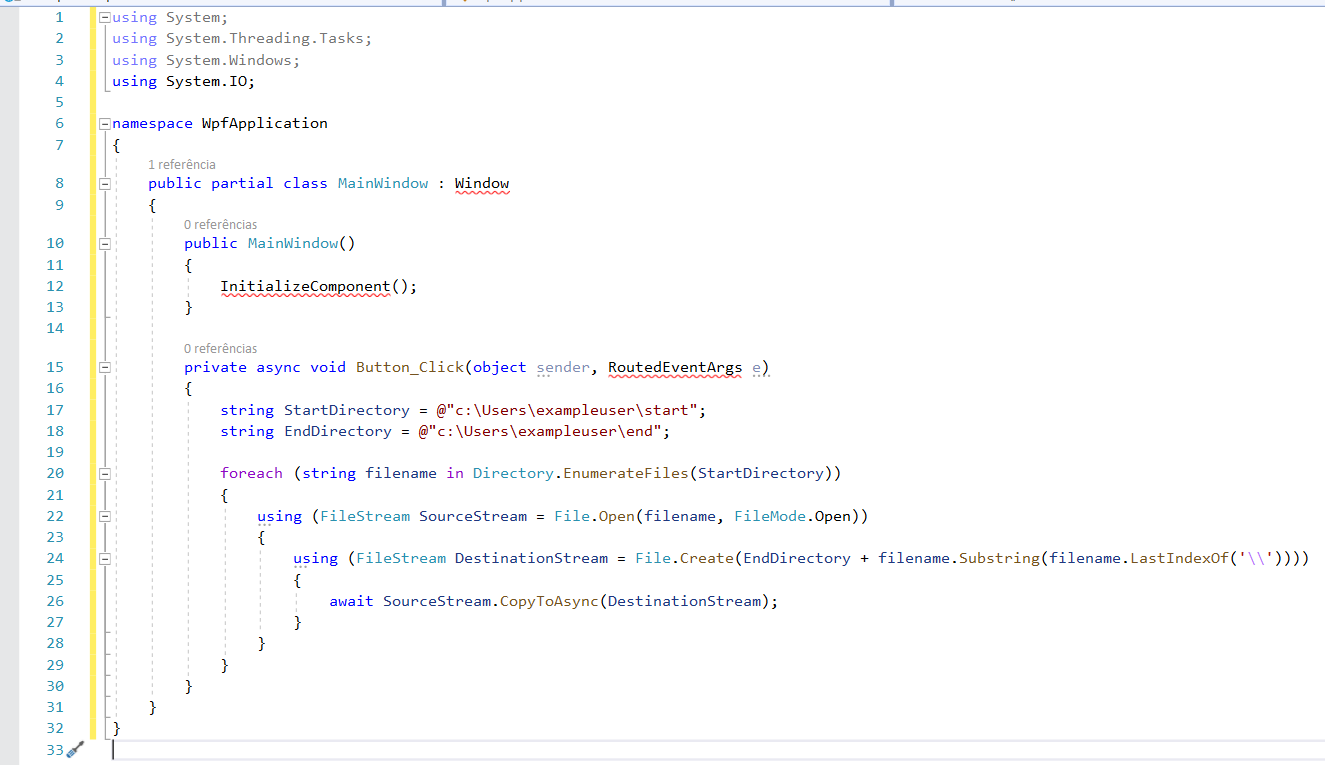
Descartar um [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) objeto libera todos os dados armazenados em buffer e, essencialmente, chama o [Flush](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.flush?view=net-5.0) método para você. [Dispose](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.dispose?view=net-5.0) o também libera recursos do sistema operacional, como identificadores de arquivos, conexões de rede ou memória usada para qualquer buffer interno. A [BufferedStream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.bufferedstream?view=net-5.0) classe fornece a capacidade de encapsular um fluxo em buffer em torno de outro fluxo a fim de melhorar o desempenho de leitura e gravação.

A partir do .NET Framework 4,5, a [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) classe inclui métodos assíncronos para simplificar operações assíncronas. Um método assíncrono contém Async em seu nome, como [ReadAsync](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.readasync?view=net-5.0) ,, [WriteAsync](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.writeasync?view=net-5.0) [CopyToAsync](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.copytoasync?view=net-5.0) e [FlushAsync](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.flushasync?view=net-5.0) . Esses métodos permitem que você execute operações de e/s com uso intensivo de recursos sem bloquear o thread principal. Essa consideração de desempenho é particularmente importante em um aplicativo de armazenamento do Windows 8. x ou aplicativo de área de trabalho em que uma operação de fluxo demorado pode bloquear o thread da interface do usuário e fazer seu aplicativo aparecer como se não estiver funcionando. Os métodos assíncronos são usados em conjunto com async as await palavras-chave e em Visual Basic e C#.

Quando usado em um aplicativo da loja do Windows 8. x, o [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) inclui dois métodos de extensão: [AsInputStream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.windowsruntimestreamextensions.asinputstream?view=net-5.0) e [AsOutputStream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.windowsruntimestreamextensions.asoutputstream?view=net-5.0) . Esses métodos convertem um [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) objeto em um fluxo no Windows Runtime. Você também pode converter um fluxo no Windows Runtime em um [Stream](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream?view=net-5.0) objeto usando os [AsStreamForRead](https://docs.microsoft.com/pt-br/dotnet/api/system.io.windowsruntimestreamextensions.asstreamforread?view=net-5.0) [AsStreamForWrite](https://docs.microsoft.com/pt-br/dotnet/api/system.io.windowsruntimestreamextensions.asstreamforwrite?view=net-5.0) métodos e. Para obter mais informações, consulte [como converter entre fluxos de .NET Framework e fluxos de Windows Runtime](https://docs.microsoft.com/pt-br/dotnet/standard/io/how-to-convert-between-dotnet-streams-and-winrt-streams)

Algumas implementações de fluxo executam o buffer local dos dados subjacentes para melhorar o desempenho. Para esses fluxos, você pode usar o [Flush](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.flush?view=net-5.0) [FlushAsync](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.flushasync?view=net-5.0) método ou para limpar os buffers internos e garantir que todos os dados tenham sido gravados na fonte de dados ou no repositório subjacente.

Se você precisar de um fluxo sem armazenamento de backup (também conhecido como um Bucket de bits), use o [Null](https://docs.microsoft.com/pt-br/dotnet/api/system.io.stream.null?view=net-5.0) campo para recuperar uma instância de um fluxo que foi projetado para essa finalidade.

Exemplo em C#:

Explicação dos comandos:

**using:** Quando se trabalha com recursos não gerenciados é necessário que você faça a desalocação de memória de forma explicita ou de forma implícita já que ela não é feita de maneira automática. Ao se usar o using ele faz isso pra você de forma implícita.

**FileStream:** É uma classe que deriva da classe abstrata Stream. A FileStream fornece um Stream (exibição genérica de uma sequência de bytes) para o arquivo, dando suporte a operações de leitura e gravação síncronas e assíncronas. Você vai usar a classe FileStream para ler, gravar, abrir e fechar arquivos em um sistema de arquivos e manipular outros identificadores do sistema operacional relacionados a arquivos, incluindo pipes, entrada padrão e saída padrão. Com isso você poderá usar os métodos Read, Write, CopyTo e Flush para executar operações síncronas. Ou os métodos ReadAsync, WriteAsync, CopyToAsync e FlushAsync para executar operações assíncronas.

. **CopyTo (Stream):** Lê os bytes do fluxo atual e os grava em outro fluxo.

**File.Open(string, FileMode):** Abre um FileStream no caminho especificado com acesso de leitura/gravação sem compartilhamento.

**File.Create(string):** Cria ou substitui um arquivo no caminho especificado.

**Directory.Exists(string):** Você coloca o caminho(path) e ele vai retornar true e false. Ele retornará true se o diretório for existente. Entretanto, caso o diretório não seja existente ou ocorrer um erro ao tentar determinar se o diretório especificado existe ele retorna false.

**Directory.CreateDirectory(string):** Cria todos os diretórios e subdiretórios no caminho especificado, a menos que eles já existam

**FileStream.Flush():** Limpa os buffers desse fluxo e faz com que todos os dados armazenados em buffer sejam gravados no arquivo.

**. Substring (int 32):** Recupera uma subcadeia de caracteres dessa instancia. A subcadeia de caracteres começa em uma posição de caractere especificado e continua até o final da cadeia de caracteres.

**[FromForm]**: Obtém valores dos campos de formulário postados. É uma das origens do model Binding.

**. LastIndexOf (char):** Relata a posição de índice com base em zero da última ocorrência de um caractere Unicode especificado nesta instância.

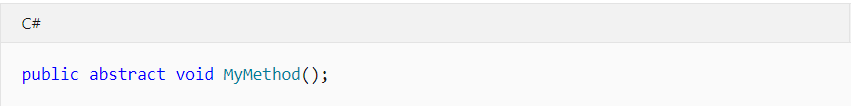
**await:** Esse operador suspende a avaliação do método async delimitador enquanto a operação assíncrona representada por seu operando não é concluída. Quando a operação assíncrona for concluída, o operador await retornará o resultado da operação, se houver. Quando o operador await é aplicado ao operando que representa uma operação já concluída, ele retorna o resultado da operação imediatamente sem a suspensão do método delimitador.

**CopyToAsync (Stream):** Lê de forma assíncrona os bytes do fluxo atual e os grava em outro fluxo.

Curiosidade:

Modificador abstract: Esse modificador indica que o item que está sendo modificado tem uma implementação ausente ou incompleta. Logo todos os membros marcados com esse modificador precisam ser implementados por classes não abstratas que derivam da classe abstrata.

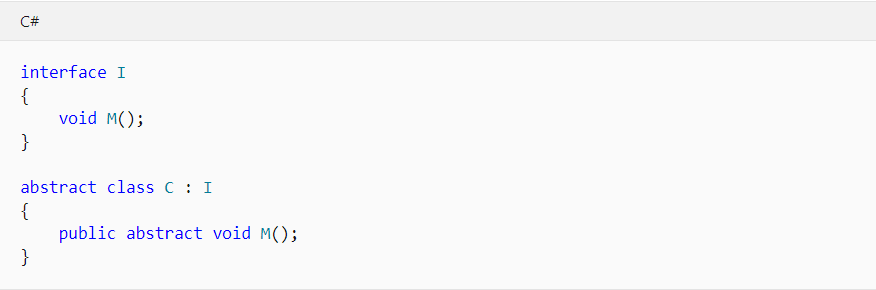
* Um método abstrato é implicitamente um método virtual.
* Declarações de método abstrato são permitidas apenas em classes abstratas.
* É um erro usar os modificadores static ou virtual em uma declaração de método abstrato.
* Como uma declaração de método abstrato não oferece nenhuma implementação real, não há nenhum corpo de método, a declaração do método simplesmente termina com um ponto e vírgula e não há chaves ({}) após a assinatura. Por exemplo:



Propriedades abstratas se comportam como métodos abstratos, exceto pelas diferenças na sintaxe de declaração e chamada.

* É um erro usar o modificador abstract em uma propriedade estática.
* Uma propriedade herdada abstrata pode ser substituída em uma classe derivada incluindo uma declaração de propriedade que usa o modificador override.

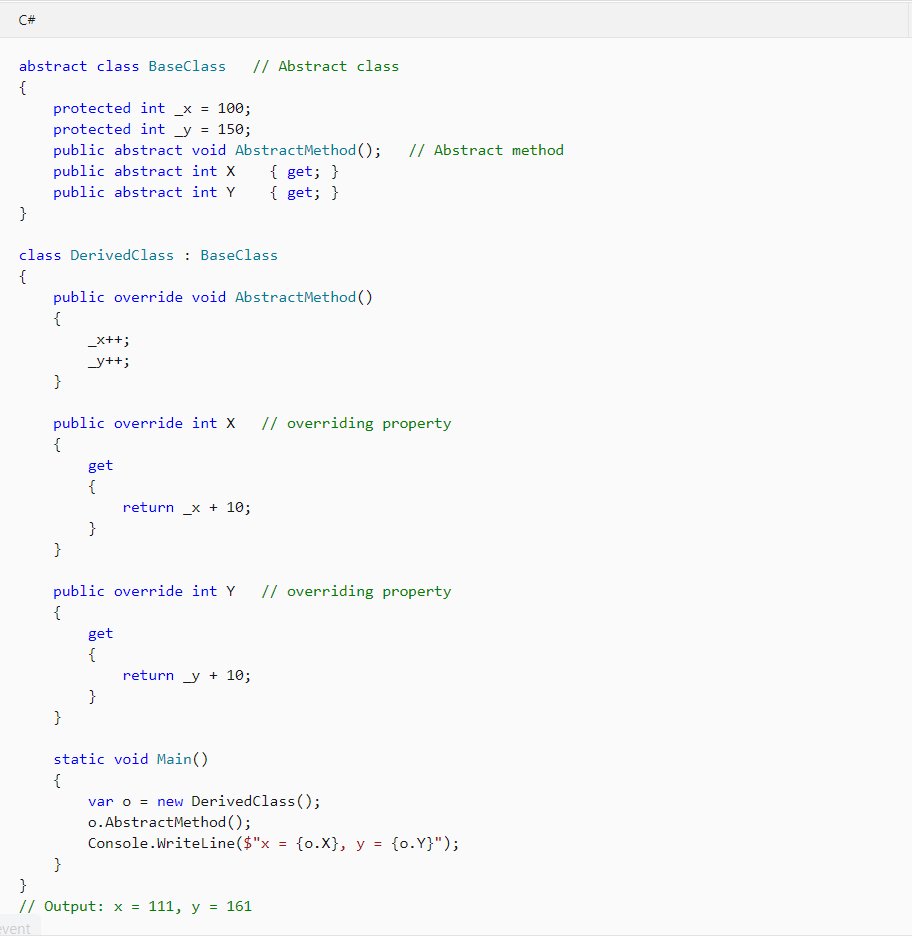
Uma classe abstrata que implementa uma interface pode mapear os métodos de interface em métodos abstratos. Por exemplo:



Classe Abstrata: Uma classe abstrata ela é destinada somente a ser uma classe base para outras classes logo ela não é instanciada por contra própria.

* Uma classe abstrata não pode ser instanciada.
* Uma classe abstrata pode conter acessadores e métodos abstratos.
* Não é possível modificar uma classe abstrata com o modificador sealed porque os dois modificadores têm significados opostos. O modificador sealed impede que uma classe seja herdada e o modificador abstract requer uma classe a ser herdada.
* Uma classe não abstrata derivada de uma classe abstrata deve incluir implementações reais de todos os acessadores e métodos abstratos herdados.

Exemplo que mostra uma classe derivada usando uma classe abstrata:



Model Binding: Os controladores e as páginas Razor funcionam com dados provenientes de solicitações HTTP. Por exemplo, dados de rota podem fornecer valores para as propriedades do modelo. O model Binding automatiza esse processo. O sistema de model Binding:

* Recupera dados de várias fontes, como dados de rota, campos de formulário e cadeias de caracteres de consulta.
* Fornece os dados para controladores e paginar Razor em parâmetros de método e propriedades públicas.
* Converte dados de cadeia de caracteres em tipos .NET.
* Atualiza as propriedades de tipos complexos.