**Common type system -** O Common type system define como os tipos são declarados, usados e gerenciados no common language runtime e também é uma parte importante do suporte do runtime para integração entre linguagens. O Common type system executa as seguintes funções:

* Estabelece uma estrutura que ajuda a permitir a integração entre linguagens, segurança de tipo e execução de código de alto desempenho.
* Fornece um modelo orientado a objetos que suporta a implementação completa de muitas linguagens de programação.
* Define regras que as linguagens devem seguir, o que ajuda a garantir que os objetos escritos em diferentes linguagens possam interagir uns com os outros.
* Fornece uma biblioteca que contém os tipos de dados primitivos (como Boolean, Byte, Char, Int32 e UInt64) usados no desenvolvimento de aplicativos.

**Global Assembly Cache(GAC)** nada mais é do que um repositório centralizado para o armazenamento de assemblies (DLLs) gerados em .NET. Cada versão do .NET Framework conta com um repositório próprio. Partindo desta premissa, toda vez que um assembly tenha sido adicionado ao GAC, o mesmo estará disponível para uso por qualquer aplicação .NET compatível com a versão da biblioteca adicionada ao repositório.

**Common Language Runtime (CLR) -** O Common Language Runtime (CLR) é o componente do .NET Framework responsável pelo gerenciamento da execução de programas gravados em qualquer uma das várias linguagens suportadas, permitindo que eles compartilhem classes comuns orientadas a objetos escritos em qualquer um dos idiomas. Podemos comparar o Common Language Runtime com a máquina virtual Java que a Sun Microsystems fornece para executar programas compilados a partir da linguagem [Java](https://celsokitamura.com.br/java/). A Microsoft chama o Common Language Runtime como um “ambiente de execução gerenciado.” Um programa compilado para o CLR não precisa de um ambiente de execução específico do idioma e pode ser movido facilmente para ser executado em qualquer sistema suportado pelo .NET Framework.

**Classes Parciais -** É possível dividir a definição de uma [classe](https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/class) ou [struct](https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/struct), uma [interface](https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/interface) ou um método em dois ou mais arquivos de origem. Cada arquivo de origem contém uma seção da definição de tipo ou método e todas as partes são combinadas quando o aplicativo é compilado.

public partial class Employee

{

public void DoWork()

{

}

}

A palavra-chave partial indica que outras partes da classe, struct ou interface podem ser definidas no namespace. Todas as partes devem usar a palavra-chave partial. Todas as partes devem estar disponíveis em tempo de compilação para formar o tipo final. Todas as partes devem ter a mesma acessibilidade, tais como public, private e assim por diante.

Na linguagem C#, a palavra reservada “**sealed**” pode ser utilizada na declaração de classes. Ela tem a função de impedir que a classe seja derivada por outras classes.

Na plataforma .NET, a classe **base** de todas as outras classes é a **System.Object.**

Na plataforma .NET, os controladores do formulário apresentado estão todos definidos no *namespace*: **System.Windows.Forms.**

Na linguagem de programação C#, para fazer o compilador ignorar a checagem estática de tipos e permitir, em tempo de compilação, que qualquer operação seja executada sobre um determinado objeto, é necessário que ele seja declarado como: **dynamic**.

O principal *namespace* da ADO.NET, que contém as classes utilizadas pelos provedores de acessos (que representam tabelas, colunas, linhas etc.) é denominado **System.Data.**

No C#, a coleção HashSet: **não** permite a inclusão de elementos duplicados. Nenhum "**Set**" permite elementos duplicados.

Em c#, métodos em uma interface só podem ser **públicos** e sem **implementação**.

Muito importante não confundir com a linguagem java, em C# não possuímos a palavra reservada implements nem extends. Ex.:

* Java - PessoaJuridica implements ITributavel
* C# - PessoaJuridica : ITributavel

No .NET Framework, fazem parte do namespace **System.Collections** as seguintes classes:

* **Hashtable:** representa uma coleção de pares chave-valor organizados com base no código hash da chave.
* **Queue:**representa uma coleção primeiro a entrar, primeiro a sair de objetos.
* **Stack:** representa uma coleção não genérica simples UEPS (último a entrar, primeiro a sair) de objetos.
* **Comparer:** compara dois objetos para equivalência, no qual as comparações de cadeias de caracteres diferenciam maiúsculas de minúsculas.

No C#, o bloco *lock* evita a ocorrência de: **condição de corrida (*race condition*).**

O 'bloco' lock, em linguagem C# é semelhante ao método synchronized, da linguagem Java, o qual funciona como um 'semáforo' garantindo acesso exclusivo a um certo recurso.

Mas por que não é alternativa B? O bloco lock também não evita deadlock? Na verdade a questão esta correta pois foi extraída daqui: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/lock-statement

Mas conforme menciona Tannenbaum, para haver um deadlock (impasse) é necessário haver 4 condições( Exclusão mútua, posse e espera, não preempção e espera circular). Condição de corrida é evitado com a exclusão mútua - acesso exclusivo - corroborando para o gabarito da questão, pois para haver deadlock é necessário existir uma condição de corrida e MAIS 3 CONDIÇÕES. Ou seja, o bloco lock evita uma *race condition*e não as outras 3 condições, por isso não podemos afirmar que o bloco lock evita deadlock

[Questões de Provas - Questões de Concursos - Página 3 | Qconcursos.com](https://www.qconcursos.com/questoes-de-concursos/questoes?discipline_ids%5B%5D=160&examining_board_ids%5B%5D=1&examining_board_ids%5B%5D=2&examining_board_ids%5B%5D=5&examining_board_ids%5B%5D=152&page=3&subject_ids%5B%5D=1292)