**Introdução**

1. O C# foi inspirado nas linguagens de programação C e C++ com influencias de Java também.
2. A linguagem de programação C# foi criada para ser utilizada com a nova arquitetura .NET criada pela Microsoft. Como sendo uma coisa completamente nova não teria preocupações com código de legado.
3. No antigo .NET Framework só era possível desenvolver através de uma plataforma Windows e para não ficar presa a apenas uma plataforma a Microsoft criou a plataforma .NET Core, cross-platform, modular e open-source. Sendo acessível o desenvolvimento em outras plataformas além de Windows.
4. .NET Framework surgiu em 2002, desenvolvimento exclusivo no Windows, composto por dois componentes CLR o mecanismo de execução que manipula os aplicativos em execução, e a biblioteca de classes .NET Framework. O .NET Core surgiu em 2016 num esforço da Microsoft para desenvolver um framework mais moderno e multiplataforma modularizada e open-source. O Core também pode ser utilizado junto com o Docker que o .NET Framework não permite.

**Utilizando POO**

1. POO é uma sigla para Programação Orientada a Objeto  
   A sua ideia é aproximar as estruturas (classes) de objetos da vida real, onde cada objeto contém seus atributos (dados atrelados a esse objeto) e métodos (ação que aquele objeto realiza).
2. Polimorfismo, quem vem do grego poli = muitas, morphos = forma, é o principio pelo qual duas ou mais classes que derivem de uma mesma classe possam invocar métodos que tem a mesma assinatura, porém com comportamentos podendo ser completamente distintos já que as classes derivadas podem personalizar o método.
3. Abstração em POO é a possibilidade de criação de classes abstratas, que servem como modelo de atributos e métodos. Sem a abstração não conseguiríamos ter o conceito do polimorfismo na POO.
4. Encapsulamento é um conceito do POO para proteger os atributos de um objeto. Em C# esse conceito pode ser implementado definindo o modificador de acesso da propriedade para Private, e exibindo para leitura e alteração através de métodos. Assim passa-se a ter controle do que é acessado e alterado dentro do objeto.
5. Devemos usar uma interface quando precisamos somente definir a assinatura de métodos e propriedades obrigatórios, não tendo nenhuma implementação nesses métodos, ficando a responsabilidade de implementar a classe que irá herda-la. Já a classe abstrata utilizamos quando precisamos de uma classe base, com métodos que podem ter ou não ter implementação, podendo a classe que herde ela subscrever ou não esses métodos.
6. Interfaces:
   1. IDisposable: As classes que herdam essa interface devem implementar o método Dispose() para liberar recursos não gerenciados pelo CLR. Assim ao realizar a chamada do método deve-se liberar a memória desse objeto. É comum a utilização em identificadores de arquivo ou objetos de conexões de banco de dados.
   2. IComparable: As classes que herdam essa interface devem implementar o método CompareTo(object), onde deve-se ser feita a comparação de um objeto com o outro. Esse método é chamado nos métodos Sort() de Array e List .
   3. ICloneable: As classes que herdam dessa interface devem implementar o método Clone(), que será criado um novo objeto a partir do já existem, criando um clone.
   4. IEnumerable: É uma interface base utilizada para realizar iterações de coleções não genéricas, contém um único método, o GetEnumerator() que retorna um enumarador que itera em uma coleção. Utilizado para habilitar o foreach do c# para percorrer a coleção.
7. Não, o C# só permite a herança de apenas uma classe abstrata, porém permite herdar de várias interfaces. Com esse comportamento de só herdar de apenas uma classe não acontece conflito de duas classes herdadas com o mesmo método com implementações concretas. Por exemplo, uma classe Carro herde de duas classes Automovel e Motocicleta que possuem implementação concreta do método Acelerar(), ao chamar o método Acelerar() do objeto Carro de qual classe abstrata terá duas implementações para serem chamadas. Esse é o problema conhecido como problema diamante  
   

**Testes de Unidade**Para testes de unidades em C# se utiliza muito os frameworks XUnit, NUnit e o framework da própria Microsoft, o MSTest, porém os mais utilizados são os XUnit e NUnit.

Para testes automatizados de tela temos o SpecFlow que podemos usar a técnica do BDD (Behaviour Driven Development) para o desenvolvimento.