C# POO

* Construtor da classe, possibilitando receber argumentos e inicializar atributos a partir da criação de um objeto;
  + Com isso, a inicialização dos atributos recebidos no construtor torna-se obrigatória.
* Atributos da classe, os atributos estáticos;
* Métodos da classe, os métodos estáticos;
* Ausência de referência, do **this**, dentro de métodos estáticos.
* **Herança:** Aprendemos herança e vimos que, com sua sintaxe bastante simples, ela é importantíssima em qualquer projeto.
* **Polimorfismo:** Com isso, foi possível tratar objetos do tipo Diretor como Funcionario e evitar repetição de código e várias sobrecargas iguais.
* **Modificadores virtual e override:** Alguns comportamentos possuem implementações diferentes nas classes derivadas, para resolver este problema basta usarmos métodos virtuais e a sobrescrita.
* **base:** A classe filha pode fazer referência aos membros da classe base com uso desta palavra reservada.
* O modificador de acesso protected;
* Preenchendo construtores da classe base a partir da classe derivada, com a sintaxe public Diretor(string cpf) : base(5000, cpf);
* Classes e métodos abstratos!

Interface x Classe abstrata

* A interface funciona como um contrato, tendo somente métodos e todos por padrão abstratos, não sendo necessário o uso da diretiva abstract.
* A Classe abstrata pode ter métodos concretos apesar de não poder ser instanciada.
* Uma determinada classe filha pode herdar tanto de uma classe abstrata, quanto uma interface
* As classes filhas da interface não necessitam das diretivas override, nem as interfaces da diretiva virtual
* Propagar erros com retorno de métodos não é correto;
* Podemos usar as exceções do .NET;
* Para tratar exceções do .NET, basta usar os blocos try/catch;
* As exceções são representadas por objetos e duas propriedades importantes são a Message e StackTrace

O StackTrace começa quando lançamos a exceção com throw <objeto de exception>;. Desta forma, se usamos esta sintaxe em blocos catch, estaremos perdendo informações da exceção original.

using = basicamente um try/catch/ finally encapsulado, desde que o finally tenha somente o objetivo de fechar instancias criadas no try/catch, implementa a interface iDispose, que tem o método Dispose, que funciona exatamente como o finally.

* O modificador de acesso internal;
* O modificador de acesso internal protected;
* Revisamos todos os outros modificadores de acesso;
* O que é o NuGet;
* O comando Install-Package;
* O tipo DateTime;
* O tipo TimeSpan;
* Conhecemos a biblioteca Humanizer.