1. [6] Pretende-se desenvolver uma biblioteca para definição e validação de regras sobre propriedades de objectos.
   Uma regra é representada pela interface `IValidation { bool Validate(object obj); }` (exemplos de implementações desta interface: `Above18`, `NotNull`).

   Uma instância de `Validator<T>`) representa um validador para uma entidade de domínio de tipo `T`, aos quais é possível acrescentar regras de validação para determinadas propriedade (identificada pelo seu nome), tal como apresentado no exemplo seguinte:

`struct Student {`
`  public int Age;`
`  public String Name;`
`}`
	`Student s = new Student(); s.Age = 20; s.Name = "Anacleto";`
	`Validator<Student> validator = new Validator<Student>()`
	`  .AddValidation("Age", new Above18())`
	`  .AddValidation("Name", new NotNull());`
	`validator.Validate(s);`

   a) [1] Implemente a classe `Above18`, que retorna `true` se o valor recebido por parâmetro for superior a 18, ou `false` caso contrário.

   b) [2] Implemente a classe `Validator<T>` tendo em conta que o método `Validate` lança a exceção `ValidationException`, se falhar alguma das regras.

   c) [2] Sem alterar o código escrito anteriormente, acrescente o necessário para que seja suportada a adição de regras na forma de delegates do tipo `Func<W, bool>`, conforme demonstra o exemplo seguinte.
   Se a propriedade indicada não for do tipo `W`, então é lançada a exceção `TypeMismatchException`.

`Validator<Student> validator =`
`  new Validator<Student>()`
`  .AddValidation<String>("Name",`
`  UtilMethods.Max50Chars);`
	`class UtilMethods {`
	`  public static bool Max50Chars(String s) {`
	`  /* ... */`
	`  }`
	`}`

   d) [2] Adicione o necessário e modifique apenas o método `AddValidation(string name, IValidation val)` de modo a que sobre a mesma propriedade possam ser adicionados vários validadores.

2. Reflexão, Interfaces, Delegates

...