Ambientes virtuais de Execução – Teste de Época Normal – 7 de Julho de 2017
2016/2017 Semestre de Verão - Duração 2h30

Número: _____ Nome: _____

Grupo 1 [10 valores]

Nas questões 1 a 4, marque cada alternativa como verdadeira (V) ou falsa (F). Uma alternativa assinalada corretamente conta 0,5 valores, incorretamente desconta 0,25 valores ao total da respectiva questão.

1. [2] No sistema de tipos da plataforma .NET
   a. ____ os tipos valor não podem redefinir métodos virtuais.
   b. ____ a expressão `Object i = 10;` não dá erro de compilação nem de execução.
   c. ____ a definição `namespace N { class C<W> : IEnumerable<T> { void M(W w, T t) { } } }` é válida.
   d. ____ a expressão `typeof(DateTime).GetType() == typeof(String).GetType()` é verdadeira.
2. [2] Considerando o delegate `Func<A, Object>`, e sabendo que a classe A é base de C, os seguintes métodos podem ser usados para criar instâncias deste tipo de delegate:
   a. ____ `Object M(A c) { ... }`
   b. ____ `String M(A a) { ... }`
   c. ____ `String M(C c) { ... }`
   d. ____ `A M(Object o) { ... }`
3. [2] O número de *bytes* ocupado por cada instância de um objecto de tipo T aumenta com
   a. ____ o número de propriedades definidas em T com implementação explícita de `get` e/ou `set`.
   b. ____ o número de variáveis locais definidas em métodos de instância de T ou das suas classes base.
   c. ____ o número de instâncias de T criadas por reflexão com sobrecarga da respectiva dimensão.
   d. ____ o número de métodos de instância definidos no tipo T e nas respectivas classes base.
4. [2] Os *custom attributes* da plataforma .NET
   a. ____ quando aplicados a campos de instância fazem aumentar o tamanho em bytes ocupado por cada instância.
   b. ____ podem ser instanciados sem ser via chamada a `Attribute.GetAttributes(...)`.
   c. ____ podem ter vários construtores definidos por delegates através da API de reflexão.
   d. ____ são sempre aplicáveis a qualquer tipo de membro.
5. [2] Complete a definição do método seguinte assegurando flexibilidade máxima nos parâmetros de tipo genérico em falta:

```
Dictionary<___, List<___>> CollectBy<_________>(List<___> src, Func<_____> toKey, Func<_____> toElem)
{
        Dictionary<___, List<___>> res = new Dictionary<___, List<___>>();
        foreach(___ item in src) {
            ___ key = toKey(item);
            List<___> elems;
            if(!res.TryGetValue(key, out elems)) { elems = new List<___>(); res[key] = elems; }
            elems.Add(toElem(item));
        }
        return res;
}
```

Grupo 2 [10 valores]

Ambientes virtuais de Execução – Teste de Época Normal – 7 de Julho de 2017
2016/2017 Semestre de Verão - Duração 2h30

6. [5] Pretende-se desenvolver uma biblioteca de regras para validar propriedades de objectos de qualquer tipo. Uma regra é representada pela interface `IValidation { bool validate(object t); }`. A classe `ValidatorBuilder` cria validadores para o tipo `T` (instâncias de `Validator<T>`), aos quais é possível acrescentar regras de validação para uma determinada 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 = ValidatorBuilder
                    .Build<Student>()
                    .AddValidation("Age", new NotUnder18())
                    .AddValidation("Name", new NotNull());
validator.Validate(s);
```

a) [1] Implemente a classe `ValidatorBuilder`. O método `Build` retorna uma nova instância de `Validator<T>`.
b) [2] Implemente a classe `Validator<T>` tendo em conta que o método `Validate` lança a exceção `ValidationException`, se falhar alguma das regras. Implemente também a classe `NotUnder18`.
c) [2] Acrescente o necessário para a biblioteca também suportar a adição de regras na forma de delegates do tipo `Func<W, bool>`, conforme demonstra o exemplo seguinte. É lançada a exceção `TypeMismatchException` se a propriedade indicada não for do tipo `W`.

```
Validator<Student> validator =
  ValidatorBuilder
     .Build<Student>()
     .AddValidation<String>("Name",
                            UtilMethods.Max50Chars);
```

```
class UtilMethods {
  public static bool Max50Chars(String s) {
    /* ... */
  }
}
```

7. [2] Apresente o código IL gerado para o método `C.Oper`: NOTA: 'a' == 97

```
class C {
   private int val;
   public C(int v) { val = v; }
   public int V { get { return val; } }
   public int Oper(String s) { return val = V + s.IndexOf('a') + 1; }
}
```

8. [1.5] Dada a seguinte definição parcial:

```
class Grupo2 { delegate void Action(int i); class A { /*...*/ } }
```

escreva em C# o equivalente ao seguinte troço de código IL:

```
.method static class Grupo2.Action Identity(class Grupo2.Action a) cil managed {
  IL_0000:  ldarg.0
  IL_0001:  brfalse.s  IL_0005
  IL_0003:  ldarg.0
  IL_0004:  ret
  IL_0005:  newobj     instance void Grupo2.A::.ctor()
  IL_000a:  ldftn      instance void Grupo2.A::M(int32)
  IL_0010:  newobj     instance void Grupo2.Action::.ctor(object, native int)
  IL_0015:  ret
}
```

brfalse.s -> Transfers control to a target instruction if *value* is **false**, a null reference, or zero.

9. [1.5] Acrescente à interface `IEnumerable<T>` suporte para a operação ***lazy*** `Collapse`, que retorna uma nova sequência que junta os elementos adjacentes iguais da sequência original (segundo o método `Equals`). Exemplo:

```
int[] a = { 7, 7, 9, 11, 11, 3, 3, 9, 9, 7 };
foreach(int i in a.Collapse())  // escreve no standard output
   Console.Write(i + "; ");      // 7; 9; 11; 3; 9; 7;
```