Ambientes virtuais de Execução – Teste de Época de Recurso – 20 de Julho de 2017
2016/2017 Semestre de Verão

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] Considere a interface `IEnumerable<T>`:
   a. ____ Não podem existir variáveis deste tipo, uma vez que todos os seus métodos são abstratos.
   b. ____ Não podem existir variáveis de instância deste tipo, uma vez que todos os seus métodos são abstratos.
   c. ____ O seu descritor de tipo nunca é retornado pelo método Object.GetType().
   d. ____ A expressão `new IEnumerable<string>();` é válida.

2. [2] No tipo genérico `MyType<T>`, sobre `T`....
   a. ____ Apenas podem ser chamados os métodos de `Object` se não for definida alguma restrição.
   b. ____ Podem ser chamados os métodos com sobrecargas redefinidas no managed heap.
   c. ____ Pode sempre ser chamado o construtor sem argumentos.
   d. ____ Podem ser aplicadas restrições para aumentar o número de métodos que podem ser chamados nesse tipo.

3. [2] Com a biblioteca `Reflection.Emit` é possível...
   a. ____ acrescentar métodos a tipos da biblioteca standard
   b. ____ acrescentar campos a tipos da biblioteca standard
   c. ____ definir classes cujo construtor usa tipos emitidos com esta biblioteca
   d. ____ definir propriedades sem método `get`

4. [2] Na linguagem IL ...
   a. ____ existe a noção de registos virtuais da máquina virtual
   b. ____ nenhuma instrução tem em conta as relações de herança
   c. ____ as variáveis locais têm tipo mas este não é usado na codificação das instruções da família `ldloc`
   d. ____ tal como para os campos, existem instruções específicas para obter o valor de propriedades

5. [2] Tendo em conta as seguintes definições:

```
interface I { void M(); void G(); }
class A : I { public virtual void M() { Console.WriteLine("A"); }
              public virtual void G() { Console.WriteLine("A::G"); } }
class B : A { public override void M() { Console.WriteLine("B"); } }
class C : B { public override void M() { Console.WriteLine("C"); }
              public new void G() { Console.WriteLine("C::G"); } }
```

a) Escreva o *output* de `((B)new C()).M();`

b) Escreva o *output* de `((A)new C()).G();`

Ambientes virtuais de Execução – Teste de Época de Recurso – 20 de Julho de 2017
2016/2017 Semestre de Verão
Grupo 2 [10 valores]

6. [5] Pretende-se desenvolver uma biblioteca para comparar propriedades de objectos de qualquer tipo T. As propriedades de T têm obrigatoriamente de verificar **uma das** seguintes hipóteses: i) implementar `IComparable`, ii) estar anotada com indicação de um tipo que implemente `IComparer` ou iii) ter associado um *delegate* que especifique o critério. Os dois últimos casos, ii) e iii), usam critérios de comparação independentes do tipo da propriedade. Nas questões das alíneas seguintes, a) cobre os casos i) e ii), enquanto b) cobre o caso iii).

   a) [3] Tendo em consideração o código seguinte, implemente a classe `DiffReporter<T>` e o atributo `CriterionAttribute`.
   O método Compare compara todas as propriedades dos dois objetos do tipo T e retorna uma sequência *lazy* de `KeyValuePair<String, int>`, onde a chave é o nome da propriedade e o valor é o resultado da comparação (< 0, = 0, > 0).
   As propriedades não anotadas têm de ser compatíveis com `IComparable`, para serem comparáveis entre si. Para as propriedades anotadas com `CriterionAttribute` é usada uma instância do tipo indicado no atributo para realizar a comparação. Este tipo tem de implementar a interface `IComparer`.

```
DiffReporter<Student> cmp = new DiffReporter<Student>();
Student s1 = new Student { Age=20, Name="Ze", City="Lisboa" };
Student s2 = new Student { Age=25, Name="Maria", City="Lisboa" };
foreach (KeyValuePair<String, int> r in cmp.Compare(s1, s2)) {
   Console.WriteLine("Prop={0} Res={1}", r.Key, r.Value);
}
```

OUTPUT:

```
Prop=Age Res=-1
Prop=Name Res=1
Prop=City Res=0
```

Tipos da plataforma .NET

```
interface IComparable
   {int CompareTo(Object);}
interface IComparer
   {int Compare(Object, Object);}
```

Tipos para o exemplo apresentado acima

```
class Student {
  public int Age { get; set; }
  [Criterion(typeof(NameCmp))]
  public String Name {get;set;}
  public String City {get;set;}
}
```

```
class NameCmp: IComparer
{
  . . .
}
```

   b) [2] Acrescente à classe `DiffReporter<T>` a possibilidade de especificar critérios de comparação na forma de *delegates* do tipo `Func<W, W, int>`, sendo W o tipo da propriedade à qual se associa o critério, tal como demonstrado no exemplo seguinte:

```
DiffReporter<Student> cmp = new DiffReporter<Student>();
cmp.For<int>("Age", (x, y) => { return y - x; /*reverse order*/ });
foreach (KeyValuePair<String, int> r in cmp.Compare(s1, s2)) {
   Console.WriteLine("Prop={0} Res={1}", r.Key, r.Value);
}
```

7. [2] Apresente o código IL gerado para o método `ConditionalAction.TryToAct`:

```
abstract class ConditionalAction {
   private readonly Action<int> action;
   public ConditionalAction(Action<int> action) { this.action = action; }
   protected abstract bool Test(int num);
   public bool TryToAct(int val) {
      bool mayAct = Test(val);
      if (mayAct) action(val);
      return mayAct;
   }
}
```

8. [3] Pretende-se estender a interface `IEnumerable<T>` para ter a operação

   `Dictionary<K, int> CountBy<T, K>(this IEnumerable<T> before, Func<T, K> keySelector)`

   que recebe uma função (`keySelector`) para extrair a chave de cada um dos objetos da sequência anterior e retorna um `Dictionary<K, int>`, onde cada entrada contém uma chave distinta, o valor a contagem do número de objetos da sequência que produziram essa chave.

   a. [2] Implemente o método `CountBy`, com o comportamento descrito.
   b. [1] Indique a linha em falta no código seguinte para que o resultado final apresentado na consola seja o indicado.

```
List<String> strs = {“a”, “aa”, “abc”, “b”, “bb”, “bba”, “bac”, “c”, “cc” };

[[ UMA LINHA DE CÓDIGO EM FALTA ]]

foreach(KeyValuePair<string, int> entry in result) {
   Console.log(“key: {0} - value: {1}”, entry.Key, entry.Value);
}

// O resultado esperado na consola é:
key: a - value: 3
key: b - value: 4
key: c - value: 2
```