**Ambientes virtuais de Execução – 2º Teste de Época Normal – 25 de Janeiro de 2018**
**2017/2018 Semestre de Inverno - Duração 2h30**

**Número: _____ Nome: _____**

Nas questões 1 a 3, 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. A instrução IL:

a) __ `callvirt` nunca é usada na chamada a um construtor.
b) __ `call` é sempre usada na chamada a métodos estáticos.
c) __ `callvirt` nunca é usada na chamada a um *delegate*.
d) __ `call` nunca é usada na chamada a métodos de instância.

2. A compilação de: `delegate void Bar();` gera:

a) __ uma classe `Bar` que estende da classe `MulticastDelegate`
b) __ uma classe `MulticastDelegate` com um método: `void Bar()`
c) __ uma classe `Bar` que herda um método: `void Invoke()`
d) __ uma classe `Bar` com um novo método: `void Invoke()`

3.

```
1.  class Sammy : Attribute { public int Nr { get; set; } }
2.  [Sammy(Nr = 71)] class Game { }
3.  [Sammy(Nr = App.Foo())] class Deal { }
4.  public class App {
5.    public static int Foo() { return 17; }
6.    static void Main() {
7.      typeof(Game).GetCustomAttribute<Sammy>().Nr = 19;
8.      Console.WriteLine(typeof(Game).GetCustomAttribute<Sammy>().Nr);
9.    }
10. }
```

Para a definição dada de `Sammy`, `Game`, `Deal` e `App`:

a) __ a instrução `da linha 3` dá erro de compilação.
b) __ a instrução `da linha 8 retorna 71.`
c) __ a instrução `da linha 7 lança uma excepção.`
d) __ a instrução `(new Sammy()).Nr = 15; dá erro de compilação.`

1. [2] Escreva em IL o código do construtor de `Whim` e do método `Fire`.

```
class Whim {
  public Whim() { Handler += Bundle; }
  public Action Handler { get; set; }
  void Fire(int nr) { Handler.Method.Invoke(this, null); }
  void Bundle(object me, object you) { }
}
```

```
delegate void Action(object a, object b);
```

2. [2] Acrescente à interface `IEnumerable<T>` suporte para a operação ***lazy*** `Mul`, que recebe uma sequência de elementos (e.g. `nrs`) e um inteiro `times` e produz uma nova sequência onde cada elemento é repetido um número de vezes igual a `times`. Exemplos:

```
IEnumerable<char> chars = "ola" ;
foreach (char c in chars.Mul(3))
  Console.Write(c); // > ooolllaaa
```

```
IEnumerable<int> nrs = new int[]{ 1, 2, 3};
foreach (int n in nrs.Mul(2))
  Console.Write(n); // > 112233
```

3. [10] A classe `Asserter<T>` permite verificar a igualdade entre objectos do mesmo tipo. Dois objectos são iguais se os valores de todos os seus **CAMPOS** forem iguais entre si segundo os seguintes critérios:
   1. **Campos** primitivos se forem iguais de acordo com o seu método `Equals`;
   2. **Campos** compatíveis com `IEnumerable` se todos os seus elementos forem iguais entre as sequências. Os critérios de igualdade entre os elementos são os mesmos três aqui enumerados.
   3. Restantes tipos são iguais se todos os seus **campos** forem iguais segundo os três critérios enumerados.

Exemplo:

```
Person ze = new Person(65124, "Ze Lopes", new Company("Juju"), new string[]{"ISEL", "Chelas", "PT"});
Person to = new Person(65124, "Ze Lopes", new Company("Juju"), new string[] { "ISEL", "Chelas", "PT" });
Asserter<Person> a = new Asserter<Person>();
a.DeepEqual(ze, to);
```

O método `DeepEqual` de `Asserter` lança a excepção `AsserterException` quando os objectos recebidos por parâmetro não forem iguais segundo os critérios enumerados.

```
public class Asserter<T> {
  readonly EqualityRef eq;
  public Asserter() { eq = new EqualityRef(...); }
  public void DeepEqual(T o1, T o2) {
    eq.AreEqual(o1, o2);
  }
  void EqualsBy(string fldName, Func<T,T, bool> p) {
    eq.EqualsBy(fldName, p);
  }
}
```

```
public interface IEquality {
  void AreEqual(object o1, object o2);
}

class EqualityRef : IEquality {
  private readonly List<IEquality> eqs;
  …
  public EqualityRef(...) {...}
  void AreEqual(object o1, object o2) {... }
}
```

Responda às questões de acordo com a especificação do enunciado.
Além dos métodos pedidos **poderá ter que implementar outras classes auxiliares**.

a) [5] Implemente o construtor de `EqualityRef` que vai preencher a lista `eqs` de modo a que o seu método `AreEqual` dê a `Asserter` o comportamento especificado.
   Implemente o método `AreEqual` que usa a lista `eqs` para verificar a igualdade entre os parâmetros `o1` e `o2`.

b) [2] De modo a permitir a extensibilidade a outros critérios de igualdade a classe `Asserter` tem um método `EqualsBy` que faz corresponder a um campo (e.g. `name`) a função responsável por verificar a igualdade. Exemplo:
   `a.EqualsBy("name", (Person p1, Person p2) => p1.name.ToLower().Equals(p2.name));`

   Implemente o método `EqualsBy` de `EqualityRef` de modo a que o seu método `AreEqual` use a função (i.e. `(Person p1, Person p2) => p1.name.ToLower().Equals(p2.name)`) na verificação de igualdade em vez do critério de igualdade por omissão. Não modifique o código feito na alínea a).

c) [3] Pretende-se que os campos de `T` possam ser anotados com um *custom attribute* que especifica o critério de igualdade entre os valores desse campo.
   Escreva o código necessário para que `EqualityRef` tenha em conta o critério de igualdade anotado nos campos.
   Implemente também o *custom attribute* e exemplifique a sua utilização para especificar o critério de igualdade em dois casos: 1) campo `name` de `Person` que compara *strings* em minúsculas e 2) campo `birthDate` de `Person` que compara apenas o ano da data de nascimento.