Ambientes virtuais de Execução – Teste de Época Normal – 11 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 tradução pelo *just-in-time compiler* (*jitter*) da instrução IL:

a) __ `callvirt` resulta sempre num despacho dinâmico.
b) __ `call` resulta sempre num despacho estático.
c) __ `newobj` inclui sempre um despacho estático na chamada ao construtor.
d) __ `newobj` inclui sempre um despacho dinâmico na chamada ao construtor se a classe tiver classes derivadas.

2. Considerando que COREINFO_CLASS_STRUCT é a estrutura com informação de tipo mantida pelo CLR para cada tipo, então uma instância de tipo valor:

a) __ **nunca tem** um cabeçalho com um ponteiro para a sua COREINFO_CLASS_STRUCT.
b) __ **só tem** um cabeçalho com um ponteiro para a sua COREINFO_CLASS_STRUCT se estiver **boxed**.
c) __ **tem** um cabeçalho com um ponteiro para a sua COREINFO_CLASS_STRUCT se estiver afecta a um campo de instância **de uma classe**.
d) __ **tem** um cabeçalho com um ponteiro para a sua COREINFO_CLASS_STRUCT se estiver afecta a um campo de instância **de uma struct**.

3.

```
class Sammy : Attribute { public int Nr { get; set; }}
class Game { [Sammy(Nr=71)] public static int count; [Sammy(Nr=67)] public int nr; … int Foo() {…}}
```

Para a definição dada de `Sammy` e `Game`:

a) __ a instrução `typeof(Game).GetField("count").GetCustomAttribute<Sammy>().Nr;` retorna **sempre** 71.
b) __ a instrução `typeof(Game).GetField("nr").GetCustomAttribute<Sammy>().Nr;` retorna 67 **só** se `Game` tiver sido instanciado, uma vez que `nr` é um campo de instância.
c) __ a utilização `[Sammy(Nr=Foo()]static int count;` **não pode** ser feita sobre o campo estático `count`.
d) __ a utilização `[Sammy(Nr=Foo()]int nr;` pode ser feita sobre o campo de instância `nr` porque o método `Foo` é de instância.

4. [2] Escreva em IL o código do construtor de `Wacky` e do método `Fire`.

```
class Wacky {
    public Wacky() { Handler = Bundle; }
    public Action Handler { get; set; }
    void Fire(int nr) { Handler(nr); }
    void Bundle(object msg) { }
}
```

```
delegate void Action(object i);
```

5. [2] Acrescente à interface `IEnumerable<T>` suporte para a operação ***lazy*** `Flatten`, que recebe uma sequência de subsequências (e.g. `words`) e junta todos os elementos de cada subsequência numa nova sequência. Exemplo:

```
IEnumerable<IEnumerable<char>> words = new string[] { "ola", "super", "isel" };
foreach(char c in words.Flatten()) Console.Write(c); // > olasuperisel
```

6. [10] A classe `Conveyor<T,R>` faz parte de uma solução de correspondência de instâncias do tipo `T` para `R`.
   Todas as propriedades de `T` com o **mesmo nome** e **tipo compatível** de uma propriedade de `R` são copiadas para uma nova instância de `R` pelo método: `R Convey(T source)`.
   A estrutura de dados `convs` mantém a correspondência entre propriedades de `T` e `R`.
   É possível corresponder propriedades de tipo compatível e **nome diferente** através do método `Match()`.
   Admita que `R` tem sempre um construtor sem parâmetros.

```
public class Conveyor<T, R> {
  /// A chave representa uma propriedade de T
  /// O valor representa o setter de uma propriedade de R
  Dictionary<PropertyInfo, ISetter> convs;

  public void Match(string from, string to) {…}
  public R Convey(T source) {…}
}
```

```
public interface ISetter {
  /// <summary>
  /// Afecta uma propriedade de
  /// target com val
  /// </summary>
  void Set(object target, object val);
}
```

Exemplo de utilização:

```
public class Student {
  public int Nr {
    get; set;
  }
  public string Name {
    get; set;
  }
  public string School{
    get; set;
  }
  …
}
```

```
public class Person {
  public int Id { get; set; }
  public string Name {get; set; }
  public Company Company{get;set;}
  …
}
public class Company {
  public string Name {get; set; }
  …
}
```

```
Student st = new Student(
    65125, "Ze Lopes", "ISEL");

Conveyor<Student, Person> conv =
    new Conveyor<Student, Person>();
conv.Match("Nr", "Id");

Person p = conv.Convey(st);

Assert.AreEqual(st.Nr, p.Id);
Assert.AreEqual(st.Name, p.Name);
Assert.AreEqual(null, p.Company);
```

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

a) [3] Implemente o construtor de `Conveyor` que vai popular `convs` e implemente o método `Match`.

b) [2] Implemente o método: `R Convey(T source)` .

c) [2] De modo a permitir a correspondência entre propriedades de tipo incompatível foi adicionado um novo método `Match` que faz corresponder uma propriedade de `T` (e.g. `School`) a uma função que afecta uma propriedade de `R` (e.g. `Company`), conforme o exemplo seguinte:

```
conv.Match("School", (Person target, string sch) => target.Company = new Company(sch));
Person p = conv.Convey(st); … Assert.AreEqual(st.School, p.Company.Name);
```

Implemente o método `Match()` sem alterar nenhum do código das alíneas anteriores.

d) [3] Pretende-se que as propriedades de `R` possam ser anotadas com um *verificador* que testa se o valor passado à propriedade é válido. Escreva o código necessário para que a solução afecte as propriedades de `R` apenas quando o valor passado for válido para o verificador anotado na propriedade.
Além da implementação, exemplifique a utilização de dois *verificadores*: um na propriedade Id que só aceita valores superiores a 50000 e outro na propriedade Name que só aceita strings de dimensão inferior a 100.