Ambientes virtuais de Execução –Teste de Época Regular– 28 de Janeiro de 2020
2019/2020 Semestre de Inverno - Duração 2h00

Número: _____ **Nome:** _____

Nas perguntas de escolha múltipla assinale a **ÚNICA resposta correcta**. Cada pergunta de escolha múltipla **errada desconta 50% da cotação da pergunta ao total do exame**, sem resposta 0 valores.

1) [0,75] A chamada ao construtor **sem** parâmetros de um tipo valor gera em IL a instrução:
   - ❑ newobj
   - ❑ call
   - ❑ initobj
   - ❑ ldftn

2) [0,75] A chamada ao construtor **com** parâmetros de um tipo valor gera em IL a instrução:
   - ❑ newobj
   - ❑ call
   - ❑ initobj
   - ❑ ldftn

3) [0,75] Dado a variável `int n = 7`, então a instrução: `object o = n;` gera em IL a instrução:
   - ❑ box
   - ❑ unbox
   - ❑ castclass
   - ❑ nenhuma das opções

4) [0,75] Dado a variável `string s = "ola"`, então a instrução: `object o = s;` gera a em IL a instrução:
   - ❑ box
   - ❑ unbox
   - ❑ castclass
   - ❑ nenhuma das opções

```
public static IEnumerable<IEnumerable<T>> Echo<T>(
  IEnumerable<T> src, int nr)
{
  List<IEnumerable<T>> res = new List<IEnumerable<T>>();
  foreach (T item in src) {
    Console.Write(item);
    res.Add(Repeat(item, nr));
  }
  return res;
}
```

```
static IEnumerable<T> Repeat<T>(
  T item,
  int nr)
{
  for (int i = 0; i < nr; i++) {
    Console.Write(item);
    yield return item;
  }
}
```

Dado `string[] src = { "a", "b", "c" }` indique o que é apresentado no *standard output* na execução de:

5) [0,75] `Echo(src, 2)`

_____

6) [0,75] `foreach (IEnumerable<string> sub in Echo(src, 2)) { }`

_____

7) [0,75] `foreach (IEnumerable<string> sub in Echo(src, 2)) { foreach (string item in sub) { } }`

_____

Considere a definição dos tipos `Supplier` e `App` e o resultado simplificado da compilação do método `Run` em IL:

```
delegate object Supplier();
class App {
  string target = "OLA";

  Supplier DoIt(Supplier task) {  return task; }

  void Run() { Console.WriteLine( DoIt(target.ToLower)() );  }
}
```

```
1  _______
2  ldarg.0
3  _______    App::target
4  _______    String::ToLower
5  _______    Supplier
6  callvirt   ******::*******
7  callvirt   ____________________
8  call       Console::WriteLine
```

8) [0,75] A instrução IL da linha 1 em falta no método Run é:
   - ❏ ldloc.1
   - ❏ ldfld
   - ❏ ldarg.0
   - ❏ ldftn

9) [0,75] A instrução IL da linha 3 em falta no método Run é:
   - ❏ ldloc.1
   - ❏ ldfld
   - ❏ ldarg.0
   - ❏ ldftn

10) [0,75] A instrução IL da linha 4 em falta no método Run é:
   - ❏ callvirt
   - ❏ ldfld
   - ❏ newobj
   - ❏ ldftn

11) [0,75] A instrução IL da linha 5 em falta no método Run é:
   - ❏ callvirt
   - ❏ ldfld
   - ❏ newobj
   - ❏ ldftn

12) [0,75] A instrução IL da linha 7 em falta no método Run é:
   - ❏ App::DoIt
   - ❏ Supplier::Invoke
   - ❏ Supplier::ToLower
   - ❏ String::ToLower

13) [11] Considere os tipos `Pipe`, `IOperation` e `IListener` que permitem o encadeamento de operações e sua execução.

```
public class Pipe<T> {
  IListener<T> lst;
  IList<IOperation<T>> ops = new List<IOperation<T>>();

  public void Listen(IListener<T> lst) { this.lst = lst; }

  public void Add(IOperation<T> op) { ops.Add(op); }

  public T Run(T prev)
  {
    foreach (IOperation<T> op in ops) {
      prev = op.Execute(prev);
      lst.OnExecute(op.GetName(), prev);
    }
    return prev;
  }
}
```

```
Public interface IOperation<T> {
  string GetName();
  T Execute(T src);
}
```

```
public interface IListener<T> {
  void OnExecute(string name, T res);
}
```

Nas respostas **pode implementar funções ou tipos auxiliares**, mas **NÃO pode modificar** as definições dadas e **nem adicionar campos** a Pipe<T>. Em cada alínea pode reutilizar, ou não, métodos das alíneas anteriores.

a) [1] Implemente as classes `ConsoleListener` e `LowerCase` que satisfazem o seguinte teste unitário com sucesso e produz o respetivo *output*. (`String` disponibiliza o método de instância `string ToLower ()`)

```
Pipe<string> pipe = new Pipe<string>();
pipe.Listen(new ConsoleListener());
pipe.Add(new RemoveSpaces());
pipe.Add(new LowerCase());
Assert.AreEqual("joanagabrielamaria",
                pipe.Run("Joana Gabriela Maria"));
```

Output:

```
RemoveSpaces --> JoanaGabrielaMaria
LowerCase --> joanagabrielamaria
```

b) [2] Faça uma nova implementação do método `Listen` que combina o *listener* existente em `Pipe<T>` (caso exista) com o novo *listener* `lst` recebido por parâmetro.

c) [2] Implemente na classe `Pipe<T>` o método de instância `Function` que adiciona ao *pipeline* uma operação baseada na função recebida por parâmetro.
E.g. dado `PipeTest`: `class PipeTest { static string Quote(string src) { return ">" + src; }}` então podemos ter: `pipe.Function(PipeTest.Quote);`

d) [2] Implemente na classe `Pipe<T>` o método de instância `Static` que adiciona ao *pipeline* uma nova operação baseada no **nome método público estático e na classe** recebidos por parâmetro.
Se o método identificado não for compatível com o *pipeline* então `Static` retorna sem alterar `Pipe<T>`.
E.g. `pipe.Static(typeof(PipeTest), "Quote");`

e) [2] Implemente na classe `Pipe<T>` o método de instância `Method` que adiciona ao *pipeline* uma nova operação baseada no **nome método público de instância e na classe** recebidos por parâmetro.
O método recebido (e.g. `ToLower` de `String`) não tem parâmetros e o seu *target* será o resultado da operação anterior no *pipeline*. (Pode implementar apenas as diferenças para a alínea d)
E.g. `pipe.Method(typeof(string), "ToLower");`

f) [2] Implemente o método estático `Pipe<V> Build<V>(Type klass)` que retorna uma nova instância de `Pipe<V>` contendo operações para os métodos **públicos** de **instância** e **estáticos** de `klass` que sejam compatíveis com o *pipeline* e anotados com um *custom attribute* definido para esse efeito (defina-o como entender).
Esse *custom attribute* deve ainda indicar a ordem que o método anotado deve ocupar no *pipeline*.