Ambientes virtuais de Execução –Teste de Época de Recurso – 13 de Fevereiro 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) [1] O número de bytes ocupado por cada instância de `class A { ... }` aumenta com
   ____ o número de campos de instância definidos em A.
   ____ o número de campos de instância e estáticos definidos em A.
   ____ o número de campos de instância definidos em A e variáveis locais definidas em métodos de instância de A.
   ____ o número de campos de instância e estáticos definidos em A e variáveis locais definidas em métodos de instância e estáticos de A.

2) [1] Os *custom attributes* da plataforma .NET
   ____ quando aplicados a campos de instância fazem aumentar o tamanho em bytes ocupado por cada instância.
   ____ podem ter vários construtores.
   ____ só podem ser aplicados a campos ou propriedades.
   ____ nenhuma das opções.

3) [1] A expressão `Object i = 10;`
   ____ dá um erro de compilação.
   ____ compila sem erros, mas dá uma excepção em execução.
   ____ compilada para IL inclui uma operação de box.
   ____ compilada para IL NÃO inclui uma operação de box.

4) [1] A compilação de: `delegate void Bar();` gera uma classe com um método:
   ____ `void Bar()`
   ____ `void Invoke()`
   ____ `void Invoke(object target, object[] args)`
   ____ nenhuma das opções.

5) [1] Considere A e B duas **estruturas (tipos valor)** definidas pelo programador. Sendo a e b duas variáveis do tipo A e B, então a compilação da instrução: `b = (B) a;` resulta:
   ____ numa operação de boxing
   ____ numa operação de unboxing
   ____ num erro de compilação
   ____ apenas numa cópia do valor da variável `a` para a variável `b`

```
class A {public virtual void Foo(){}} class App {void Main(){A a = new A(); a.Foo();}}
```
A instrução `a.Foo()` compilada para IL gera as duas instruções: `ldloc.0  callvirt instance void A::Foo()`

6) [1] Alterando a definição de `Foo` para `public void Foo(){}` o compilador gera:
   ___ `call instance void A::Foo()`
   ___ `ldnull  call instance void A::Foo()`
   ___ `ldloc.0  call instance void A::Foo()`
   ___ `ldloc.0  callvirt instance void A::Foo()`

7) [1] Alterando a definição de `Foo` para `public static void Foo(){}` o compilador gera:
   ___ `call void A::Foo()`
   ___ `ldnull  call void A::Foo()`
   ___ `ldloc.0  call void A::Foo()`
   ___ `ldloc.0  callvirt void A::Foo()`

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

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

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

8) [0,5] `Echo(src, 2)`

_____

9) [0,5] `foreach (IEnumerable<string> sub in Echo(src, 2)) { }`

_____

10) [0,5] `foreach (IEnumerable<string> sub in Echo(src, 2)) { foreach (string item in sub) { } }`

_____

Considere a definição dos tipos `Collector`, `Container` e `App` e o resultado da compilação do método `Main` em IL:

```
delegate int Collector(object o);
class Container {
  static ArrayList lst= new ArrayList();
  public static ArrayList Bag { get { return lst; } }
}
class App {
  static Collector Load(Collector handler) { return handler; }
  static void Main() { Load(Container.Bag.Add)("ola"); }
}
```

```
1  _______    _______::_________
2  dup
3  _______    _______::_________
4  _______    _______::_________
5  _______    _______::_________
6   ldstr      "ola"
7  _______    _______::_________
```

Complete TODOS os espaços _____ da:

11) [0,5] linha 1: _____ _____::_____

12) [0,5] linha 3: _____ _____::_____

13) [0,5] linha 4: _____ _____::_____

14) [0,5] linha 5: _____ _____::_____

15) [0,5] linha 7: _____ _____::_____

16) [9] A classe `ComplexCmp<T>` permite comparar instâncias de `T` com base nos seus campos comparáveis (compatíveis com `IComparable`)

```
public class ComplexCmp<T> {
  List<IComparator> cmps = new List<IComparator>();

  public ComplexCmp() { ... }

  public int Compare(object x, object y) {
    foreach (IComparator c in cmps) {
      int res = c.Compare(x, y);
      if (res != 0) return res;
    }
    return 0;
  }
}
```

```
public interface IComparable {
   int CompareTo(object obj);
}
```

```
public interface IComparator {
   int Compare(object x, object y);
}
```

Considere o seguinte caso de utilização da classe `Student` com os **campos públicos** `nr` do tipo `int`, `name` do tipo `string` e `addr` do tipo `Address` que não é comparável (i.e. `Address` não implementa `IComparable`):

```
Student s1 = new Student(14000, "Ana", new Address("Rua Amarela", 24));
Student s2 = new Student(14000, "João", new Address("Rua Rosa", 30));
Student s3 = new Student(11000, "João", new Address("Rua Rosa", 16));
Student s4 = new Student(11000, "João", new Address("Rua Verde", 48));

ComplexCmp<Student> cmp = new ComplexCmp<Student>();
int res1 = cmp.Compare(s1, s2); // res1 < 0 porque a string Ana precede a string João
int res2 = cmp.Compare(s2, s3); // res2 > 0 porque 14000 é maior que 11000
int res3 = cmp.Compare(s3, s4); // res3 = 0 porque todos os campos IComparable são iguais
```

Nas respostas **pode implementar funções ou tipos auxiliares**, mas **NÃO pode modificar** as definições dadas e **nem adicionar campos** a `ComplexCmp<T>`.
São penalizadas respostas com implementações que agravem a eficiência!

a) [3] Sem modificar NADA da classe `ComplexCmp<T>` **implemente o construtor** que vai preencher a lista `cmps` de modo a que o seu método `Compare` tenha o comportamento especificado.

b) [3] Implemente em `ComplexCmp<T>` o método `Add` que permite adicionar outros critérios de comparação na forma do exemplo seguinte (e.g. campo `addr` do tipo `Address`):

```
ComplexCmp<Student> cmp = new ComplexCmp<Student>();
cmp.Add((std1, std2) => std1.addr.nr - std2.addr.nr);
int res4 = cmp.Compare(s3, s4); // res4 < 0 porque 16 é menor que 48.
```

c) [3] Pretende-se que a classe identificada por T possa ser anotada com *custom attributes* (e.g `ComparatorStudentAccountByBalance`, `ComparatorStudentAdressByStreet`, entre outros) que permitam definir outros critérios de comparação a incluir no algoritmo definido por `ComplexCmp<T>`, conforme o exemplo seguinte:

```
[ComparatorStudentAccountByBalance()]
[ComparatorStudentAdressByStreet()]
public class Student { ... }
```

```
class ComparatorStudentAccountByBalance : _______ {
  public override int Compare(object x, object y) {
    Student s1 = (Student)x;
    Student s2 = (Student)y;
    return s1.acc.balance - s2.acc.balance;
  }
}
```

```
class ComparatorStudentAdressByStreet : _________ {
  public override int Compare(object x, object y) {
    Student s1 = (Student)x;
    Student s2 = (Student)y;
    return s1.addr.street.CompareTo(s2.addr.street);
  }
}
```

Complete e Implemente o tipo base das classes usadas no exemplo: `ComparatorStudentAccountByBalance` e `ComparatorStudentAdressByStreet`.

Implemente o código que adicionaria ao construtor de `ComplexCmp` da alínea a) para que passe a incluir os critérios anotados no algoritmo de comparação.