Ambientes virtuais de Execução – 2º Teste de Época Normal – 6 de Julho de 2018
2017/2018 Semestre de Verão - 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)
```
interface IBean{ void Set(int v); int Get();}
struct S : IBean {
      public int x;
      public void Set(int v) { this.x = v; }
      public int Get() { return this.x; }
}
```
Dadas as variáveis s e i definidas pelas expressões: `S s = new S(); IBean i = s;`

a) ___ A execução de `((IBean) s).Set(11); Console.WriteLine(s.x);` tem o *output* 11.

b) ___ A execução de `((S) i).Set(11); Console.WriteLine(i.Get());` tem o *output* 11.

c) ___ A execução de `s.x = 11; Console.WriteLine(i.Get());` tem o *output* 11.

d) ___ A execução de `s.Set(11); Console.WriteLine(i.Get());` tem o *output* 11.

2)

`class A {`
`  int x;`
`  int y;`
`  int z;`
`}`
	`class B {`
	`  int X{ get{ return 7; } set{} }`
	`  int Y{ get{ return 11; } set{} }`
	`  int Z{ get{ return 19; } set{} }`
	`}`
		`class C {`
		`  static int x;`
		`  static int y;`
		`  static int z;`
		`}`

a) ___ uma instância de `A` ocupa o mesmo espaço em memória que uma instância de `B`.

b) ___ uma instância de `A` ocupa o mesmo espaço em memória que uma instância de `C`.

c) ___ uma instância de `B` ocupa o mesmo espaço em memória que uma instância de `C`.

d) ___ adicionando à definição da classe `A`: `public void Foo(){};` aumentará o espaço ocupado por uma instância de A.

3) `class A {public virtual void Foo(){}} class App {void Main(){A a = new A(); a.Foo();}}`
Para a instrução `a.Foo()` o compilador de C# gera: ldloc.0 callvirt instance void A::Foo()
Alterando a definição de `Foo` para:

a) ___ `public void Foo(){}` o compilador gera: ldloc.0 call instance void A::Foo()

b) ___ `public void Foo(){}` o compilador gera: call instance void A::Foo()

c) ___ `public static void Foo(){}` e a.Foo(); alterada para A.Foo(); o compilador gera: ldloc.0 call void A::Foo()

d) ___ `public static void Foo(){}` e a.Foo(); alterada para A.Foo(); o compilador gera: call void A::Foo()

3. [9] A classe **ObjectCopier** cria cópias de objectos do tipo referência indicado no seu construtor. O exemplo de código seguinte define os tipos **Student** e **AddressInfo** e cria uma cópia do objecto **Student** referido por s1.

```
Student s1 = new Student {
   Number = 20001, Name = "Manuel",
   Address = new AddressInfo {
      Street = "Rua do Mar",
      Door = 1
   }
};
```

```
ObjectCopier copier = new ObjectCopier(typeof(Student));
Student s2 = (Student)copier.CreateAndCopy(s1);
Console.WriteLine(
       s1.Address.Door == s2.Address.Door   // true
);
Console.WriteLine(Object.ReferenceEquals(
                  s1.Address, s2.Address));  // false
```

```
class Student {
   public int Number { get; set; }
   public String Name { get; set; }
   public AddressInfo Address { get; set; }
```

```
class AddressInfo {
   public String Street { get; set; }
   public int Door { get; set; }
}
```

O processo de cópia consiste na criação de um novo objecto do tipo em causa e na cópia das propriedades entre o objecto original e o novo. Por simplicidade, assuma que as propriedades são de tipo valor, *string* ou de um tipo referência que tem as mesmas restrições (ex: **AddressInfo**).

A cópia das propriedades é feita de uma das seguintes formas:

a) superficialmente, no caso dos tipos primitivos e *strings*, ou seja, é copiado o valor ou a referência para *string*;
b) em profundidade, para outros tipos referência, ou seja, é usado o **ObjectCopier** para criar uma cópia do objecto referido pela propriedade (note o caso de **AddressInfo** no exemplo anterior).

A interface `IPropertyCopier` representa o contracto para copiar uma determinada propriedade do objecto fonte para o objecto `destino`.
No troço de código seguinte, o construtor de **ObjectCopier**, determina, para cada propriedade, o tipo concreto que vai ser usado pelo método **CreateAndCopy** para copiar as propriedades de **source** para o novo objecto.

```
class ObjectCopier {
  Type t;
  List<IPropertyCopier> copiers = new List<IPropertyCopier>();

  public ObjectCopier(Type t)  {
    this.t = t;
    PropertyInfo[] props = t.GetProperties();
    foreach (PropertyInfo p in props) {
      if (p.PropertyType.IsValueType || p.PropertyType == typeof(String))
         copiers.Add(new PrimitiveCopier(p));
      else
         copiers.Add(new RefCopier(p));
   }
 }
 public object CreateAndCopy(object source) { /* b) */ }
}
```

```
public interface IPropertyCopier {
  void copy(object target, object source);
  PropertyInfo GetProperty();
}
```

A **eficiência** da solução implementada é contabilizada na avaliação das questões.
NÃO poderá acrescentar outros campos à classe `ObjectCopier` além dos definidos na listagem apresentada.

a) [2] Defina as classes **PrimitiveCopier** e **RefCopier**.

b) [2] Implemente o método **CreateAndCopy**. Assuma que o tipo do objecto tem construtor sem parâmetros e que **source** é do tipo representado pelo **Type** passado ao construtor.

c) [1] Sem escrever código nem dar detalhes de implementação **enumere apenas** os campos e/ou métodos da classe **`ObjectCopier`** que teria de modificar ou adicionar para suportar propriedades do tipo **`IEnumerable`**?

d) Pretende-se que antes da cópia das propriedades de tipo valor ou string, seja analisado o seu valor no objecto fonte para verificar se a cópia deve prosseguir para o objecto destino ou não.
NOTA: esta alínea não tem implicações na implementação das classes **`PrimitiveCopier`** e **`RefCopier`** da alínea a) nem de **`CreateAndCopy`** da alínea b), pelo que devem permanecer **inalteradas** e **SEM** dependências dos tipos definidos nesta questão.

1. [2] Dê suporte a esta funcionalidade através do uso de uma anotação a qual especifica o critério de validação a aplicar ao valor obtido da propriedade do objecto fonte. Apresente o código (i) da anotação (ii) das modificações na classe **`ObjectCopier`** (iii) de um exemplo de utilização para só copiar os números de aluno maiores que 0.

2. [2] Dê suporte a esta funcionalidade com um novo método que modifica a forma de copiar determinada propriedade, tendo em conta um critério na forma de um *delegate*, como demonstra a seguinte utilização:

```
copier.SetConditionalCopier<int>("Number", n => n > 0);
```

1. [2,5] Escreva em IL o código do construtor e do método e `CallHandlers` da classe `Del`.

```
delegate int Func(object obj);
class Listener {
    public int M1(object o) { return 1; }
}
```

```
public class Del {
    private Func handlers;
    public Del() { handlers = new Listener().M1; }
    public int CallHandlers() { return handlers(10); }
```

2. [2,5] Acrescente à interface `IEnumerable` suporte para a operação ***lazy* genérica** `GroupJoin`, que recebe duas sequências *fonte* (do tipo T1 e do tipo T2), e três funções: as duas primeiras funções são usadas para obter as chaves dos objetos das sequências fonte, respetivamente; a terceira função produz um objeto do tipo R (a incluir na sequência retornada pelo método `GroupJoin`) que associa *um elemento* da primeira sequência fonte com *uma sequência de elementos* da segunda sequência fonte (os elementos associados têm o mesmo valor da chave usando o método `Equals`).
Assuma que a primeira sequência fonte (T1) não contém elementos com o valor da chave repetido.
Reescreva a definição do método `GroupJoin` e `Join` completando os espaços de modo a obedecer aos requisitos e comportamento do exemplo seguinte:

```
string[] words = { "ola", "isel", "super", "nap", "gap", "katty", "loop" };
string[] sizes = { "2: ", "3: ", "4: ", "5: ", "6: " };
IEnumerable<string> res = sizes.GroupJoin(
   words,
   size => Int32.Parse(size.Substring(0,1)),
   w => w.Length,
   (string size, IEnumerable<string> ws) => size + String.Join(",", ws));
foreach (string item in res) Console.WriteLine(item);
```

```
Standard Output:

2:
3: ola,nap,gap
4: isel,loop
5: super,katty
6:
```

```
static IEnumerable<R> GroupJoin<T1, T2, K, R>(______ s1,______s2,______k1,______k2,____group) {
   foreach (T1 item in s1) {
      _______________ = Join(______, ______, ______);
      ____________ group(______, ______);
   }
}
static ________ Join<K, T2>(______, ______, ______) {
   foreach(_____________) {
      if (_________ )
          _________________;
   }
}
```