Instituto Superior de Engenharia de Lisboa
Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Ambientes Virtuais de Execução

1º Teste – versão A

6 de Julho de 2011
Duração 2h30m

Nome:_____ Número_____

Grupo I

Assinale a alternativa correcta que completa cada uma das frases. Cada frase assinalada com uma alternativa incorrecta desconta metade da cotação da frase ao total do grupo.

1. [1] Uma instância que está na *freachable queue* …
   [ ] tem a garantia que o finalizador corre pelo menos uma vez.
   [ ] tem a garantia que os finalizadores das instâncias por si referenciadas correm antes do seu.
   [ ] tem a garantia que pode aceder aos objectos por si referenciados, pois estes estão atingíveis.
   [ ] não tem nenhuma das garantias anteriores.

2. [1] Numa interface em C# podem não podem ser declarados delegates, porque…
   [ ] o delegate teria de ser abstracto, uma vez que a interface também o é.
   [ ] não seria possível ao runtime implementar o construtor e os métodos de Invoke, uma vez que a interface é abstracta.
   [ ] apenas podem ser declarados membros de comportamento numa interface.
   [ ] apenas podem ser declarados métodos numa interface.

3. [1] Uma das razões que justifiquem a escrita de código específico para controlar a subscrição/revogação de eventos em C# é…?
   [ ] … permitir que um tipo tenha mais que um evento do mesmo tipo.
   [ ] … evitar a utilização de MulticastDelegates substituindo-os por Delegates.
   [ ] … optimizar o espaço ocupado em memória das instâncias dos tipos que definem eventos.
   [ ] ... suportar cenários onde o tipo a que pertence o evento é genérico e o tipo do evento também é genérico em pelo menos um dos tipos que parametrizam do tipo englobante.

4. [1] Sem o suporte da linguagem C# para definir restrições nos tipos que parametrizam tipos e/ou métodos genéricos…
   [ ]… seria possível criar instâncias dos argumentos de tipo invocando um construtor com parâmetros.
   [ ]… apenas seria possível aceder à interface de Object dos argumentos de tipo.
   [ ]… seria possível aceder a toda a interface dos argumentos de tipo. As restrições servem apenas para limitar esse acesso.
   [ ]... nenhuma das opções anteriores é válida.

Grupo II

Nota: Responda, justificadamente a todas as questões.

1. [1] Qual a necessidade de criar *assemblies* multi-módulo? Apresente pelo menos dois exemplos onde esta necessidade seja evidente.
2. [2] O código seguinte é a codificação em IL de um método genérico parametrizado pelo tipo `T` que, recebendo como parâmetro um *array* `a` e um elemento `elem`, retorna o número de ocorrências de elem em `v`. Descreva o código apresentado e faça a implementação equivalente em C#.

```
.method public hidebysig static int32  Ocurrences<T>(!!T[] a,!!T elem) cil managed {
  .locals init ([0] class Exame1.Program/'<>c__DisplayClass1`1'<!!T> 'CS$<>8__locals2')
  newobj     instance void class Exame1Ocurrences.Program/'<>c__DisplayClass1`1'<!!T>::.ctor()
  stloc.0
  ldloc.0
  ldarg.1
  stfld      !0 class Exame1Ocurrences.Program/'<>c__DisplayClass1`1'<!!T>::elem
  ldloc.0
  ldc.i4.0
  stfld      int32 class Exame1Ocurrences.Program/'<>c__DisplayClass1`1'<!!T>::ocurr
  ldarg.0
  ldloc.0
  ldftn      instance void class Exame1Ocurrences.Program/'<>c__DisplayClass1`1'<!!T>::'<Ocurrences1>b__0'(!0)
  newobj     instance void class [mscorlib]System.Action`1<!!T>::.ctor(object, native int)
  call       void [mscorlib]System.Array::ForEach<!!0>(!!0[], class [mscorlib]System.Action`1<!!0>)
  ldloc.0
  ldfld      int32 class Exame1Ocurrences.Program/'<>c__DisplayClass1`1'<!!T>::ocurr
  ret
}
```

3. [3] O método `Zip` junta cada um dos elementos da primeira sequência com o elemento de mesmo índice da segunda sequência, Se as sequências não têm o mesmo número de elementos, o método junta as sequências até uma delas terminar. Este método recebe uma função que define como a junção é realizada.
   a. [1,5] Implemente o método `Zip` como um método de extensão para `IEnumerable<T>`. O método tem a seguinte assinatura:

```
IEnumerable<TResult> Zip<TFirst, TSecond, TResult>(this IEnumerable<TFirst> first,IEnumerable<TSecond> second,
    Func<TFirst, TSecond, TResult> resultSelector
);
```

   b. [1,5] Utilizando o *extension method* `Zip`, desenvolvido na alínea anterior, implemente o método `IEnumerable<string> GetFileLines(string fileName)` que retorna um enumerado de *strings*, cada um delas contendo o número de linha e o conteúdo dessa linha do ficheiro de texto cuja *path* é `fileName`.

Grupo III

1. [3] Pretende-se desenvolver um sistema de anotação que identifica métodos constantes, isto é, métodos que, quando invocados sobre um objecto, não alteram o seu estado.
   a. [0,5] Implemente o *custom attribute* `ReadOnlyAttribute`, o qual poderá ser aplicado uma vez a métodos e recebe no construtor uma `string`, correspondente à descrição do método anotado. O método ToString da classe `ReadOnlyAttribute` deverá retornar a descrição do método ao qual foi aplicado o atributo.
   b. [1,5] Implemente o método estático `IEnumerable<MethodInfo> GetReadOnlyMethods(Type type)` que retorna todos os métodos definidos para o tipo `type` que tenham sido anotados com o atributo `ReadOnlyAttribute`.
      NOTA: Valorizam-se soluções que não impliquem a criação de contentores auxiliares
   c. [1] Implemente o método `void Apply( object instance )` como um método de extensão para `object`, que invoca sobre `instance` todos os métodos de instância definidos para o tipo `de instance` que sejam constantes e sem parâmetros.

2. [4] Considere a definição dos tipos `MyInterface`, `Sucessor1`, `Sucessor2`, `DoubleSucessor` e a sua utilização na classe `Test`.

```
public interface MyInterface<T>{
  void print();
  T Get();
  void Join(T t);
}

public struct Sucessor1 : MyInterface<int>{
  protected int i;
  public void print(){
    Console.WriteLine("Sucessor = " + i);
  }
  public int Get(){ i++; return i;}

  public void Join(int j){ i+=j; j+=i; }
  public override string ToString() {
     return i.ToString();
   }
}

 public class Sucessor2 : MyInterface<int>{
   protected int i;
   public void print(){
     Console.WriteLine("Sucessor = " + i);
   }
   public int Get(){ i++; return i; }

   public void Join(int j){ i += j; j += i; }
  }
```

```
public class DoubleSucessor : Sucessor2{
 public DoubleSuccessor() { i = 2; }
 public virtual int Get(){ i=2*i; return i; }
}

class Test{
  static void Main(string[] args){
    Sucessor1 s = new Sucessor1();
    s.Get();
    s.print();
    Object o = s;
    ((Sucessor1)o).Get();
    s.print();
    s.Get();
    MyInterface<int> i = s;
    ((Sucessor1)i).Get();
    i.print();
    Console.WriteLine(s);
    Console.WriteLine(((Sucessor1)i).ToString());

    Sucessor2 d = new DoubleSucessor();
    d.Get();
    Sucessor2 e=new Sucessor2();
    int k= d.Get();
    e.Join(k);
    e.print();
    Console.WriteLine(k);
  }
}
```

   a. [3] Diga e justifique qual o *output* resultante da execução do método `Main` na classe `Test`, indentificando as operações de *boxing* e *unboxing*.

   b. [1] Considere a nova definição do método `Join – void Join(ref int j)`. Que alterações seriam necessárias nas restantes classes? Irá existir alguma diferença no *output*? Justifique.

3. [3] Analise o seguinte código. Apresente de forma que considere conveniente o estado do heap (raízes, heaps das gerações 0,1 e 2, e *Finalization* e *Freachable queues*) relativo aos objectos criados durante a execução do método `Main` nos pontos identificados pelos comentários. Considere que o código apresentado foi compilado em modo *release* e que não ocorrem ciclos de recolha para além das chamadas ao método `Collect` da classe `GC`.

```
public class B{
  private A[] array;
  public B(){ array=new A[3];}

  public A this[int index]{
     get{
       if (index < 0) return null;
       else return array[index];
     }
     set{
        if (!(index < 0)) array[index]= value;
     }
  }
}
public class A{
    public A link;
    public A() { }
    public A(A a) { link = a; }
    ~A() { }
 }
```

```
class Program{

  static void Main( ){
    B b = new B();
    A a0 = new A();
    b[0]=a0;
    b[1] = new A(a0);
    b[2]= new A(b[1]);
    A some1 = b[1];
    A some2 = b[2];
    //1
    GC.Collect();
    //2
    GC.WaitForPendingFinalizers();
    //3
    Console.WriteLine(b[0]);
  }
}
```