**Ambientes Virtuais de Execução**

**(1.º S 2010/2011 Semestre Verão)**

**Lista de Exercícios de Preparação para a 1ª Ficha**

**I Parte**

1. ***Introdução à infra-estrutura .NET***
   1. Portabilidade e independência às linguagens são características das infra-estruturas de  
      suporte à construção de software por componentes, nomeadamente do CLR. Explique sucintamente como o CLR dá suporte a cada uma dessas características.
   2. Faça uma comparação entre a utilização de um componente binário (assembly) desenvolvido para a plataforma.NET e um componente binário desenvolvido em C, nas fases de criação do componente dependente e da sua execução.
   3. Indique uma utilização da metadata presente em módulos *managed* efectuada por cada uma das seguintes entidades: compilador da linguagem C#; *loader* da máquina virtual; compilador JIT da máquina virtual.
   4. Que informação consta no manifesto de um assembly?
   5. Porque motivo os módulos não incluem manifesto?
   6. Descreva o conteúdo das tabelas TypeDef e TypeRef.
   7. O CLR suporta compilação “just-in-time”. Descreva sumariamente o que acontece na primeira vez que um método é invocado e o que acontece nas invocações seguintes (Baseie a sua explicação num tipo e em métodos por si definidos).
   8. Comente a seguinte afirmação: “A existência de um sistema de tipos comum (CTS) implica que estes tenham que ser incluídos no conjunto de tipos primitivos das linguagens de programação para a plataforma.NET”.
   9. Para que um componente seja interoperável, apenas deve incluir mecanismos suportados pelo “Common Language Specification” (CLS)?
   10. Explique sucintamente o modelo de execução virtual .NET. Nesta explicação inclua a descrição do Evaluation Stack e do Activation Record.
2. ***Conceitos fundamentais do sistema de tipos***
   1. Explique a diferença entre coerção (8.3.2. *coercion*) e conversão (8.2.1 *casting*), tal como especificado na norma *Common Language Infrastructure* (CLI).
   2. Seja uma instância v1 do tipo valor V, que redefine ToString. É indiferente usar as seguintes versões de código para afixar uma instância na consola: Console.WriteLine(v1) e Console.WriteLine(v1.ToString())?
   3. A chamada a métodos não virtuais de um tipo valor nunca têm operação de boxing?
   4. Comente a seguinte afirmação: “*A operação de unboxing não é exactamente o oposto da operação de boxing*”.
   5. Qual o tempo de vida de uma instância de um tipo valor?
   6. Uma instância de ValueType pode estar localizada em *Managed Heap*? Justifique.
   7. O método estático bool System.Object.ReferenceEquals(object, object) determina se duas referências referem o mesmo objecto. Não é essa a função do operador == quando aplicado a referências? Justifique a resposta.
   8. Considere a seguinte definição do tipo Ponto. Se o tipo Ponto fosse definido como uma classe o código IL do construtor sofreria alguma alteração? E o código nativo? Justifique.

public struct Ponto {

public int x, y;

public Ponto(int x, int y) { this.x = x; this.y = y; }

…  
}

* 1. Considere o código em C, à frente, definidos em duas unidades de compilação distintas, A e B. Diga, justificando, se é necessário recompilar o módulo B se à estrutura T for retirado o comentário (1) e o módulo A for entretanto recompilado. O comportamento seria idêntico caso se tratasse de código equivalente escrito em C#?

|  |
| --- |
| **Excerto de types.h** |
| typedef struct t {  double val;  /\* char \*name; \*/ (1)  /\* … \*/  } T; |

|  |
| --- |
| **Excerto do Módulo A** |
| #include “types.h”  …  T\* createT() { return (T \*) malloc(sizeof(T)); } |

|  |
| --- |
| **Excerto do Módulo B** |
| #include “types.h”  …  T \*t1 = createT();  …  double d = t1->val; |

* 1. Quais as diferenças entre definir uma class e uma struct?

1. ***Estrutura de tipos (construtores e membros, métodos virtuais, atributo new e tabelas de métodos)***
   1. Qual a motivação do construtor de tipo? Explique as duas políticas que determinam o momento da chamada a este construtor.
   2. Na chamada a métodos não virtuais, o compilador de C# gera instruções call ou callvirt. Explique em que situação é gerada cada uma das instruções e justifique essa diferença.
   3. Explique, usando um exemplo concreto, a relação entre o atributo new da linguagem C# e o atributo newslot da linguagem IL.
   4. Comente a seguinte afirmação: “*A selecção do método numa chamada não virtual é definida em tempo de compilação porque este é definido pelo tipo da expressão que prefixa a chamada ao método. Por sua vez, a selecção do método numa chamada virtual, apenas é definida em tempo de execução porque esta selecção depende do tipo concreto do objecto referenciado pela expressão que prefixa a chamada ao método*”.
   5. Comente a seguinte afirmação: “*Em .NET a passagem de parâmetros é sempre por valor*”.
   6. Considere a seguinte expressão em C#: E.m(1, “slb”); em que E é uma expressão. Indique a análise que o compilador tem que fazer na metadata para gerar as instruções correspondentes a esta expressão.

**II Parte**

1. Analise o seguinte tipo:

|  |
| --- |
| using System;  public struct V{  private int i;    public static bool operator==(V v1, V v2){ return v1.Equals(v2); }    public static bool operator!=(V v1, V v2){ return !(v1==v2); }    public V(int i) { this.i=i; }    public override string ToString( ){ return i.ToString(); }    public void Inc( ){ i++;}  }    class Program{  public static void Main(){  V v=new V(1);  Object o = v;  ((V) o).Inc();  System.Console.WriteLine(v);  System.Console.WriteLine( ((V) o).ToString());  System.Console.WriteLine( v == (V) o);  }  } |

* 1. Indique quais as operações de *box* e *unbox* realizadas no método **Main**, justificando-as.
  2. Indique e justifique os valores apresentados na consola.

1. Considere o seguinte troço de código:

|  |  |
| --- | --- |
| interface ICounter { void Increment(); }  struct Counter: ICounter {  int value;  public override string ToString() {  return value.ToString();  }  public void Increment() { value++; }  } | class Program {  static void Main() {  Counter x = new Counter();  Console.WriteLine(x);  x.Increment();  Console.WriteLine(x);  ((ICounter)x).Increment();  Console.WriteLine(x);  }  } |

* 1. Indique, justificando, qual a saída resultante da execução.
  2. Acrescente no final do método Main a instrução “Console.WriteLine(x.ToString())”.
     + 1. Analise o código IL produzido, em particular o prefixo “constrained. Counter” (Norma CLI, parte III, secção 2.1).
       2. Altere o código IL de forma a que seja produzido o mesmo resultado mas sem a utilização do prefixo constrained.
  3. Altere a categoria do tipo Counter para referência e explique, sucintamente, as diferenças no código IL produzido comparando com a alínea a).

1. Considere os excertos da implementação de Equals nas class B1, B2 e B3.

class B1 {

public override bool Equals(object o) {

B1 b1 = o as B1;

if (b1 == null) return false;

return true;

}

}

class B2 {

public override bool Equals(object o) {

if (o == null || o.GetType() != typeof(B2)) return false;

return true;

}

}

class B3 {

public override bool Equals(object o) {

if (o == null || o.GetType() != GetType()) return false;

return true;

}

}

* 1. Qual o único que se encontra correcto? Justifique. (Sugestão: tenha em conta a existência de classes derivadas destas.)
  2. A implementação escolhida estaria correcta se existisse uma classe base que redefinia o método Equals?

1. Consideres a redefinição do método bool Equals(object o) em tipos .NET.
   1. Quando se redefine o método Equals, deve-se redefinir também o método GetHashCode. Justifique.
   2. Porque motivo devem ser sempre criadas duas versões deste método, sendo esta boa prática ainda mais relevante em ValueTypes?
   3. Quando se redefine o comportamento de igualdade e desigualdade, não seria suficiente redefinir os operadores == e != em alternativa a redefinir o método Equals?
   4. Que cuidados adicionaistem a redefinição do método Equals em ReferenceTypes, relativamente aos ValueTypes?
2. Considere a definição da classe Program em CIL.
   1. Indique e justifique (descrevendo o que faz cada um dos métodos), o *output* resultante da execução do programa.
   2. Escreva um programa equivalente em C#.

|  |  |
| --- | --- |
| ble | ***Stack Transition:*** …, value1, value2 🡪 …  The ble instruction transfers control to *target* if value1 is less than or equal to value2. |
| br | ***Stack Transition:*** … 🡪 …  Unconditional jump to target |
| ldarg.0  ldarg.1 | ***Stack Transition:*** … 🡪 …, value  The ldarg.0 instruction pushes onto the evaluation stack, first incoming argument |
| ldc.i4.s N  ldc.i4.N | ***Stack Transition:*** … 🡪 …, num  Pushes the integer value of N onto the evaluation stack as an int32. |
| ceq | ***Stack Transition:*** …, value1, value2 …, result  Push 1 (of type int32) if value1 equals value2, else push 0. |
| div | ***Stack Transition:*** …, value1, value2 …, result  Divide two values to return a quotient or floating-point result. |
| rem | ***Stack Transition:*** …, value1, value2 …, result  Remainder when dividing one value by another. |
| add | ***Stack Transition:*** …, value1, value2 …, result  Add two values, returning a new value. |

.class public auto ansi beforefieldinit Program

extends [mscorlib]System.Object

{

.method public hidebysig static int32 Alinea4(int32 n) cil managed

{

// Code size 35 (0x23)

.maxstack 3

.locals init (int32 V\_0,

bool V\_1)

IL\_0000: nop

IL\_0001: ldarg.0

IL\_0002: ldc.i4.0

IL\_0003: ceq

IL\_0005: ldc.i4.0

IL\_0006: ceq

IL\_0008: stloc.1

IL\_0009: ldloc.1

IL\_000a: brtrue.s IL\_0010

IL\_000c: ldc.i4.0

IL\_000d: stloc.0

IL\_000e: br.s IL\_0021

IL\_0010: ldarg.0

IL\_0011: ldc.i4.s 10

IL\_0013: rem

IL\_0014: ldarg.0

IL\_0015: ldc.i4.s 10

IL\_0017: div

IL\_0018: call int32 Program::Alinea4(int32)

IL\_001d: add

IL\_001e: stloc.0

IL\_001f: br.s IL\_0021

IL\_0021: ldloc.0

IL\_0022: ret

} // end of method Program::Alinea4

.method public hidebysig static void Main() cil managed

{

.entrypoint

// Code size 20 (0x14)

.maxstack 1

.locals init (int32 V\_0)

IL\_0000: nop

IL\_0001: ldc.i4 0x4d2

IL\_0006: call int32 Program::Alinea4(int32)

IL\_000b: stloc.0

IL\_000c: ldloc.0

IL\_000d: call void [mscorlib]System.Console::WriteLine(int32)

IL\_0012: nop

IL\_0013: ret

} // end of method Program::Main