Ambientes Virtuais de Execução
(2ºS 2010/2011)

Lista de Exercícios de Preparação para a 2ª Ficha

A. Estrutura de Tipos (Passagem de parâmetros por valor e referência. Propriedades. *Interfaces*, *Arrays* e Enumerados.)

1. Considere que tem duas interfaces `TemperatureCelcius` e `TemperatureFahrenheit`. Ambas as *interfaces* têm um método `double GetTemperature( )`, mas com retornos diferentes. Será possível definir uma classe que implemente ambas as *interfaces*? Justifique.

2. Considere o seguinte código.

```
interface I{
  void visit(Visitor v);
  void print();
}
struct A: I{
  int n;
  public void incN(int inc) {
     n += inc;
  }
  public void visit(Visitor v) {
    v.update(ref this);
  }
  public void print() {
    Console.WriteLine("A =" + n);
  }
}
```

```
interface Visitor{ void update(ref A a); }

class R1: Visitor{
   public void update(ref A a){a.incN(1);}
}
class R2: R1{
   public virtual void update(ref A a) {
        a.incN(2);
   }
}
class R3: R2, Visitor{
   public override void update(ref A a){
      a.incN(31);
   }
   void Visitor.update(ref A a){a.incN(32);}
}

class Test{
   static void Main(){
      A a = new A(); a.incN(1); a.print();
      a.visit(new R1()); a.print();
      I i = a; ((A) i).incN(1); i.print();
      i.visit(new R1()); i.print();
      i.visit(new R2()); i.print();
      i.visit(new R3()); i.print();
   }
}
```

   a) Diga e justifique qual o *output* resultante da execução do método `Main` da classe `Test`, identificando as operações de *boxing* e *unboxing*.

   b) Considere a nova definição da interface `Visitor`

   ```
   interface Visitor{void update(ref I i);}
   ```

   Ao recompilar a estrutura `A` para usar a nova interface `Visitor`, diga e justifique se é necessária, ou não, alguma alteração à implementação do método `visit`, para que compile com sucesso. Em caso afirmativo indique as modificações a efectuar.

   c) Considerando que a assinatura de *update* é actualizada em cada uma das classes `R1`, `R2` e `R3` em conformidade com a nova definição da *interface* `I` e de acordo com as actualizações realizadas, ou não, à estrutura `A` no ponto anterior, indique e justifique as diferenças no *output* em relação ao cenário da alínea a.

3. Implemente o método

   `IEnumerable<T> BiggerThan( IEnumerable<T> enum, T element )  where T:Comparable<T>`

   que retorna a sequência de elementos maiores do que `element`. Altere este método de modo a que seja um método de extensão para `IEnumerable<T>`.

4. Qual a responsabilidade do construtor de tipo no CLR? Diga quais são as duas semânticas de invocação do construtor de tipo.
5. Porque motivo os projectistas da framework .NET 2.0 definiram que a interface genérica `IEnumerable<T>` estende a interface não genérica `IEnumerable`. De que forma um contentor pode implementar ambas as interfaces genérica `IEnumerator<T>` e não genérica `IEnumerator`? Realize um troço de código demonstrativo.

B. Delegates, Eventos Iteradores

1. Considere a seguinte definição:

```
public delegate void SomeEventHandler(object o, EventArgs e);
```

   a. Indique qual é o código gerado pelo compilador para a definição anterior. Nota: Na descrição omita a explicação dos métodos `BeginInvoke` e `EndInvoke`.
   b. A definição deste *delegate* segue um padrão *EventHandler*, aconselhado como uma boa prática na definição de eventos em .NET e ao qual a maioria dos tipos da Framework.NET que expõem eventos utiliza. Explique sucintamente este padrão e quais as responsabilidades de cada um dos seus intervenientes.
   c. Tendo em consideração a definição do *delegate* `SomeEventHandler`, indique o código gerado pela definição seguinte, na classe onde esta é incluída.

```
public event SomeEventHandler SomeEvent;
```

2. Considere o seguinte código:

```
interface ITemperatureAlarm {
  event EventHandler TooHot;
  void triggerHot();
}

class Sensor1 : ITemperatureAlarm {
  public event EventHandler TooHot;
  public virtual void triggerHot() {
    Console.WriteLine("Sensor1");
    if (TooHot != null)
      TooHot(this, EventArgs.Empty);
  }
}

class Sensor2 : Sensor1,ITemperatureAlarm {
  public virtual event EventHandler TooHot;
  public new void triggerHot() {
     Console.WriteLine("Sensor2");
     if (TooHot != null)
       TooHot(this, EventArgs.Empty);
  }
}
```

```
class Program {
  static void tooHot(
           object src, EventArgs args){
     Console.WriteLine("in hot 1!");
  }
  static void tooHot2(
           object src,EventArgs args) {
     Console.WriteLine("in hot 2!");
  }

  public static void Main() {
    Sensor2 s2 = new Sensor2();
    ITemperatureAlarm si = s2;
    Sensor1 s1 = (Sensor1)si;
    si.TooHot += tooHot;
    s1.TooHot += tooHot;
    s2.TooHot += tooHot2;
    si.triggerHot();
    s1.triggerHot();
    s2.triggerHot();
  }
}
```

   a. Usando o ildasm verifica-se que a assinatura (simplificada) do método de registo no evento `TooHot` na classe `Sensor1` é:

```
newslot virtual final instance void add_TooHot(class [mscorlib]System.EventHandler 'value')
```

   enquanto que no evento definido na classe `Sensor2` é:

```
newslot virtual instance void add_TooHot(class [mscorlib]System.EventHandler 'value')
```

   Justifique as diferenças observadas na assinatura dos dois métodos.
   b. Indique, justificando, qual a saída resultante da execução do código apresentado.

3. Indique, justificando, o output do seguinte troço de código:

```
class Program{
     public static void M1(int v) { Console.WriteLine(v); }
     public static void M2(int v) { Console.WriteLine(2*v); }
     static void Main(string[] args){
       Action<int> a = new Action<int>(M1);
       Action<int> b = new Action<int>(M2);
       a += b; b = new Action<int>(M1);
       a(5);
      }
}
```

4. Pretende-se implementar a classe `FibonacciSequence` que representa uma sequência enumerável dos valores da série de Fibonacci. Cada Enumerador retornado por instâncias desta classe retorna o próximo valor da série por cada iteração. A listagem seguinte apresente um exemplo de utilização desta classe bem como o respectivo *output*.

```
public static void Main() {
   FibonacciSequence fs = new FibonacciSequence();
   IEnumerator<int> fsEnum = fs.GetEnumerator();
   for (int i = 0; i < 10; i++) {
      fsEnum.MoveNext();
      Console.WriteLine("F'({0})' = {1}", i, fsEnum.Current);
   }
}
```

```
F'(0)' = 0
F'(1)' = 1
F'(2)' = 1
F'(3)' = 2
F'(4)' = 3
F'(5)' = 5
F'(6)' = 8
F'(7)' = 13
F'(8)' = 21
F'(9)' = 34
```

   a. Implemente a classe `FibonacciSequence` sem o suporte para enumeradores, existente na linguagem C#.
   b. Implemente a classe `FibonacciSequence` utilizando o suporte para enumeradores existente na linguagem C#.
   c. Que cuidados deve ser tidos em consideração, no código de utilização de instâncias desta classe?

5. Indique, justificando, o problema existente no código seguinte:

```
class TargetMethods {
     delegate void Action(int x);
     delegate void OtherAction(int x);
     public static void DoAction(Action a, int i) { a(i); }
     public static void SomeAction(int i) { Console.WriteLine(i);}

     public static void Test() {
         OtherAction o = SomeAction;
         DoAction(SomeAction, 10);
         DoAction(o, 10);
     }
  }
```

C. Genéricos, Métodos de extensão

1. Considere o seguinte troço de código C#. Qual o *output* resultante da invocação do método `f`?

```
class GenericClass<T>{
   public static int aField=1;
   public static void f( ){
      GenericClass<int>.aField=10;
      Console.WriteLine(GenericClass<int>.aField + GenericClass<char>.aField);
   }
}
```

2. Complete o método `Accumulate`, que recebe como argumentos uma sequência de elementos, um elemento *seed* e uma função acumuladora, retornando a sequência resultante da aplicação da função acumuladora.

```
IEnumerable<T> Accumulate<T>(IEnumerable<T> source, T seed, Func<T,T,T> acum)
```

   a. Justifique o motivo do *delegate* `acum` ter 3 parâmetros de tipo.
   b. Usando o método anterior, elabore um programa de teste que dado um *array* de inteiros, obtém a sequência que representa acumulação dos valores do *array*.

3. Assuma que se pretende obter uma implementação da abordagem *map/reduce* para processamento de dados.

   a. Implemente os métodos de extensão `Mapper` e `CountWords`. O método `Mapper` recebe uma colecção de enumeráveis, aos quais se pretende aplicar a operação `map`. O resultado da operação `map` a cada item é um enumerável de pares (chave, valor). Implemente o método `CountWords`, em que cada item é uma linha de texto e em que o *output* são pares, constituídos pela palavra e número respectivo de ocorrências.

```
public class Program {

public static IEnumerable<KeyValuePair<K, U>> Mapper<T, K, U>(
                this IEnumerable<T> input,
                Func<T, IEnumerable<KeyValuePair<K, U>>> map){
   //. . .
}

public static
       IEnumerable<KeyValuePair<string, int>> CountWords(IEnumerable<string> lines) {
   //…
}

public static void Main() {
  string[] lines = {"ola mundo ola ave", "ave ola ave"};
  foreach (KeyValuePair<string, int> p in CountWords(lines))
   Console.WriteLine(p);
  }
}
```

Output:

```
[ola, 2]
[mundo, 1]
[ave, 1]
[ave, 2]
[ola, 1]
```

b. Implemente os métodos `Joiner` e `Reducer`, cuja assinatura é apresentada na listagem seguinte. O método `Joiner`, recebe uma sequência de pares chave valor e retorna uma sequência de grupos, em que cada grupo inclui os elementos da sequência de entrada com chaves iguais. O método `Reducer` recebe uma sequência de grupos e aplica o delegate `reduce`, a cada um dos elementos da sequência de entrada. O delegate `reduce` retorna, para cada grupo, um par chave valor, em que o valor é determinado a partir dos elementos no grupo.

```
static class Program {

//…
public static IEnumerable<IGrouping<K, KeyValuePair<K,U>>> Joiner<K, U>(
          this IEnumerable<KeyValuePair<K, U>> input) where K:IComparable<K> {

          //…
   }

public static IEnumerable<KeyValuePair<K, V>> Reducer<K, U, V>(
     this IEnumerable<IGrouping<K, KeyValuePair<K,U>>> groups,
     Func<IGrouping<K, KeyValuePair<K,U>>, KeyValuePair<K, V>> reduce) {
//…
}
//…
}
```

c. Utilize os métodos `Joiner` e `Reducer` para estender o método `CountWords` por forma a que retorne agora o número de ocorrências de cada palavra na colecção de linhas inicial.

4. Considere a seguinte classe `G`. O tipo parâmetro da classe `G` pode ser um tipo da categoria valor ou referência. A chamada ao método virtual `ToString` terá de ser feita de forma diferente em função da categoria do tipo `T`. Explique sucintamente a solução existente na plataforma .NET para que o compilador C# consiga traduzir este código para IL.

```
class G<T> {
   private T t;
   public String m() { return t.ToString(); }
}
```