Ambientes Virtuais de Execução
(2ºS 2010/2011)

Lista de Exercícios de Preparação para a 3ª Ficha

A. Gestão de memória e *Garbage Collection*

1. O processo de recolha de memória (*garbage collection*) utiliza informação que lhe permite determinar as raízes existentes na aplicação para um dado ciclo de recolha. De que informação se trata e quem a produz?

2. Durante a execução de uma determinada aplicação .Net constata-se que os objectos que precisam de ser finalizados sobrevivem às várias execuções do GC, apesar de não serem atingíveis por nenhuma raíz. Apresente uma explicação para este comportamento.

3. Considere o código apresentado em seguida.

```
class C{
  A a;
  public C(){ a = new A();}
  public A GetA{ get{ return a;}}
  }

class A {
  B b;
  public A() { b = new B(); }
  ~A() { }
}

class B { }
```

```
class Program {
 static void Main( ) {
   C[] c  = new C[3];
   for(int i=0;i<c.Length;i++) c[i]=new C();
   A a1= c[0].GetA;
   A a2= c[1].GetA;
   // 1
   GC.Collect();
   // 2
   GC.WaitForPendingFinalizers();
   GC.Collect();
   //3
   Console.WriteLine(a1);
  }
}
```

   a) Apresente na forma que considere conveniente o estado do *heap* relativo aos objectos criados durante a execução do método `Main` nos pontos identificados pelos comentários. Considere que o código foi compilado em modo *release* e que não ocorrem ciclos de recolha para além das chamadas ao método `Collect` da classe `GC`.
   b) Altere o código anterior para que a classe `A` passe a ser do tipo `IDisposable`. Neste caso o facto de na classe `Program`, a instrução `A a2=c[1].GetA` ser substituída por
   `using( A a2=c[1].GetA){Console.WriteLine(a2);},`
   implicará alterações ao estado do *heap*? Justifique.

4. Considere o código apresentado em seguida. Apresente na 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` (o qual realiza a recolha em todas as gerações).

```
struct Entry<Key, Val> where Val : class {
  public Key key;
  public WeakReference value;
  public Entry(Key key, Val val) {
    this.key = key;
   this.value = new WeakReference(val);
  }
}

class WeakRefCache<Key, Val> where Val : class{
  private Entry<Key, Val>[] cache;
  private int size;
  public WeakRefCache() {
    cache = new Entry<Key, Val>[3];
    size = 0;
  }

  public void Add(Key key, Val val) {
    cache[size++]=new Entry<Key, Val>(key, val);
       }
  }

  class E { ~E() { } }
```

```
class Program {

   static WeakRefCache<string, E> wrcache;
   wrcache = new WeakRefCache<string, E>();

   static void Main(string[] args) {
     wrcache.Add("1", new E());
     E r2 = new E();
     wrcache.Add("2", r2);
     wrcache.Add("3", new E());
     //1
     GC.Collect();
     //2
     GC.WaitForPendingFinalizers();
     GC.Collect();
     //3
     Console.WriteLine(r2.GetHashCode());
   }
}
```

B. Reflexão

1. Para obter a instância de `Type` que descreve um tipo, em que situações deve usar o operador `typeof` e/ou o método `Object.GetType()`

2. Na classe `MemberInfo` da API de reflexão o tipo de membro é definido pela propriedade `MemberType`, que retorna um valor do Enumerado `MemberTypes`. Neste enumerado não existe forma de saber se um membro é um `delegate`. Crie o *extension method* `IsDelegate()` para `MemberInfo` que retorna `true` se o membro for um `delagate` e `false` caso contrário.

3. Implemente os métodos estáticos da classe `TypeUtils` (não considere tipos genéricos):

   a) `bool IsSubclassOf(Type t1, Type t2)` que retorna `true` se o tipo representado por t1 for derivado do tipo representado por t2.

   b) `bool IsAssignableFrom(Type tvar, Type tval)` que retorna `true` se uma variável do tipo representado por tvar pode ser afectada com um valor do tipo representado por tval.

4. Pretende-se desenvolver um sistema de controlo de erros sobre determinados tipos e métodos. O sistema assume que tanto os tipos como os métodos a testar estão anotados com o atributo `BugFixedAttribute`.

   a) Implemente o atributo `BugFixedAttribute` que pode ser utilizado na declaração de classes, estruturas e métodos, o qual poderá ser aplicado múltiplas vezes. Deverão existir dois construtores na classe `BugFixedAttribute`: um que recebe o número de erro (um `int`) e uma descrição do erro (uma `string`), e outro que recebe apenas a descrição. Quando o número de erro não é dado, é utilizado como código de erro o valor -1. O método `ToString()` da classe `BugFixedAtribute` deverá retornar o código de erro e a descrição do mesmo, se o código for positivo, ou apenas a descrição, caso contrário.

   b) Implemente o método estático `void ListAllFixedBugs(Type type)` que lista todos os erros solucionados para os métodos públicos do tipo `type`.

5. Pretende-se implementar a classe `ReflectionInvoker`. Esta classe recebe no método estático `Invoker` uma `string` com a seguinte sintaxe: <TypeName> <MethodName> {-<argumentName>=<argumentValue>}*, onde:
   - `<TypeName>` - é o nome do tipo a instanciar que deve ter um construtor público sem parâmetros, se o método a invocar não for de tipo;
   - `<MethodName>` - Nome do método a invocar;

- `{-<argumentName>=<argumentValue>}*` - Lista de pares nome valor com o nome do argumento do método e o respectivo valor.

O método retorna o objecto retornado pelo método, caso este tenha tipo de retorno ou `null`, caso contrário. Apenas são suportados tipos primitivos. Para converter `strings` para valores de tipos primitivos use a classe `Convert`.

O excerto de código seguinte representa uma aplicação (`InvokerApp.exe`) consola que recebe uma `string` no formato suportado por `ReflectionInvoker.Invoke` e apresenta na consola o objecto retornado por esse método, caso exista e um exemplo da sua utilização.

```
class Program {
    public static void Main(String args) {
          object res = ReflectionInvoker.Invoke(args);
Console.WriteLine(args);
    }
}

InvokerApp System.Console WriteLine "-value=Invoking Console.WriteLine dynamically with one int argument {0}" -arg0=1
InvokerApp System.Object GethashCode
```

C. Deployment e Assemblies

1. Para além de um nome único no contexto de uma organização, apresente outra motivação para que um *assembly* tenha *strong name*?
2. Considere dois *assemblies*, A e B, sendo que o *assembly* B tem *strong name*. Justifique se, para cada um dos seguintes cenários, é necessário que o *assembly* A tenha *strong name*:
   a) *Assembly* A usa o *assembly* B;
   b) *Assembly* B usa o *assembly* A.
3. Considere o componente C a qual usa o componente D, ambos com *strong name*. Onde está guardada a qual a versão do componente D que C quer usar?
4. Explique sucintamente de que forma é possível que determinada aplicação utilize a nova versão de um componente com *strong name*, sem que a aplicação tenha de ser modificada.
5. Considere os *assemblies* apresentados em anexo (prog.zip).
   c) Execute a aplicação `client`. Justifique a excepção apresentada.
   d) Apresente uma solução em que passa a ser usada apenas a versão 2.0.0.0 do *assembly* `math.dll`.
   e) Apresente uma solução em que continuam a ser usadas as versões referidas nos manifestos dos *assemblies* `client.exe` e `statisticutils.dll`.