Ambientes virtuais de Execução – 1º Teste de Época Normal – 19 de Junho 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) Considere a definição `struct S { int w; String b; }` e a expressão `S x = new S();` ...
   a) ___ A expressão `Console.WriteLine(x)` resulta numa operação de *box*;
   b) ___ A variável `x` refere um objecto que está alojado no *heap* se `x` for uma variável local de um método;
   c) ___ O espaço ocupado pela variável `x` pode pertencer ao *heap*;
   d) ___ Se a expressão seguinte for `x.b.Length` haverá erro de compilação;

2) Considere a definição `delegate int D(object o)` e a expressão `D dm = MyClass.M;`
   A variável `dm` ...
   a) ___ refere um objecto sem *htype*;
   b) ___ é do tipo `MethodInfo`, em tempo de execução;
   c) ___ permite a utilização `dm.Target.Invoke(null);`
   d) ___ pode vir a referir `null`;

3) Com as classes do espaço de nomes `Reflection.Emit` ...
   a) ___ é possível acrescentar métodos a classes já existentes
   b) ___ o tipo de retorno dos métodos gerados fica definido em tempo de execução do código gerador
   c) ___ as classes geradas podem derivar de 1 ou mais classes e implementar 1 ou mais interfaces
   d) ___ o código gerado pode não chegar a ser executado pelo programa gerador

1. [2,5] Escreva em IL o código do construtor de `Del` e do método `Main`.

```
delegate void Func(object o, int i);
class Del {
    private ArrayList handlers;
    public Del(Func func)  {
         handlers = new ArrayList();
         handlers.Add(func);
    }
    public Func GetHandler(int idx) {
        return (Func) handlers[idx];
    }
}
class MainClass {
    private static void M(object o, int i) { … }
    public static void Main()  {
        Del dels = new Del(M);
        dels.GetHandler(0)(null, 10);
    }
}
```

2. [2,5] Acrescente à interface `IEnumerable<T>` suporte para a operação ***lazy* genérica** `GroupBy`, que recebe uma sequência de `T` e produz uma nova sequência de sequências de `T` (i.e. `IEnumerable<IEnumerable<T>>`) que agrupa elementos com a mesma chave de acordo com a função passada a `GroupBy` (e.g. `w => w.Length`).

```
IEnumerable<string> words = new string[]
           { "isel", "k", "ola", "vue", "z", "soup", "kat", "bart" };
IEnumerable<IEnumerable<string>> res = words.GroupBy(w => w.Length);
foreach (IEnumerable<string> w in res)
     Console.WriteLine(String.Join(",", w));
}
```

```
Output:

isel,soup,bart
k,z
ola,vue,kat
```

3. [9]

```
public class Query {
   private readonly string connStr;
   private readonly Type entity;
   private readonly string pk;
   private string sql;
   readonly List<IPropertySetter> props;

   public Query(string connStr , Type entity, string table, string pk) {
          this.connStr = connStr ;
          this.entity = entity;
          this.pk = pk;
          this.props = new List<IPropertySetter>();
          foreach (PropertyInfo  p in entity.GetProperties())
          {
                if (p.PropertyType.IsPrimitive || p.PropertyType == typeof(string))
                      props.Add(new PropertyPrimitive(p));
                else
                      props.Add(new PropertyRef(p, connStr));
          }
          sql = "SELECT "
                + String.Join(", ", props.Select(p => p.GetColumn()))
                + " FROM " + table;
   }

   public object GetById(object id) { ... }
   public IEnumerable GetAll() { ... }
   public static DbDataReader Execute(string sql, string connStr) { ... }
}
```

Uma instância da classe `Query` permite executar um comando SQL sobre uma base de dados disponibilizando 2 métodos para o efeito: `IEnumerable GetAll()` e `object GetById(object id)`.
O construtor de `Query` recebe uma *connection string*, uma entidade de domínio, o nome da tabela na base de dados e o nome da coluna *primary key.*
Exemplo: `Query prodQuery = new Query(NORTHWIND, typeof(Product), "Products", "ProductID");`

Admita que:

- O nome de cada propriedade da entidade corresponde ao nome de uma coluna da tabela.
- As entidades de domínio têm sempre um construtor sem parâmetros.
- Se uma propriedade não for de tipo primitivo nem *string* então é do tipo de outra entidade de domínio, estando nesse caso anotada com informação do nome da tabela e coluna *primary key* (e.g. `Category`).

```
public class Product {
  public int ProductID { get; set; }
  public string ProductName { get; set; }
  [Query("Categories", "CategoryId")]
  public Category Category { get; set; }
  public double UnitPrice { get; set; }
}
```

Admita a existência de um método auxiliar `static DbDataReader Execute(string sql, string connStr)` que executa a *query* `sql` recebida por parâmetro retornando um `DbDataReader` com o resultado.
Ignore o tratamento e libertação de recursos sobre a base de dados.

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

a) [2] Defina a interface `IPropertySetter` com os métodos que entender necessários para responder às questões e implemente as classes `PropertyPrimitive` e `PropertyRef`.

b) [2] Implemente os métodos `GetById()` e `GetAll()` da classe `Query`. Use uma implementação *lazy* onde faça sentido.

c) [3] No caso de propriedades de tipo primitivo ou `String` pretende-se que estas possam ser marcadas com uma anotação que especifica a conversão a aplicar ao valor obtido da BD antes de afectar essa propriedade.
Considere, por exemplo, que para a propriedade `ProductName` queria converter o valor obtido da BD para maiúsculas, ou que para a propriedade `UnitPrice` queria arredondar o valor obtido da BD à unidade.

b.1) Implemente o *Custom Attribute* e demonstre como é usado no exemplo dado para as propriedades `ProductName` e `UnitPrice`.

b.2) Implemente a solução alterando **apenas** o construtor da classe `Query` e se necessário criando nova(s) classe(s) auxiliar(es).

d) [2] Com o mesmo objectivo da alínea c) pretende-se que o conversor possa ser adicionado a uma instância de `Query` através de um método `Add` que recebe uma instância de um *delegate* como no seguinte exemplo:

```
prodQuery.Add<String>("ProductName", val => val.ToUpper());
```

Implemente o método `Add`, e se necessário uma nova classe auxiliar, mas sem alterar NADA das alíneas a), b) e c).