Ambientes Virtuais de Execução
(2ºS 2010/2011)
Enunciado do Trabalho Prático

Objectivos

Prática de utilização dos mecanismos estudados na disciplina, nomedamente delegates/eventos e reflexão. Utilização de APIs fluentes.

Descrição

Pretende-se a realização de um sistema que permita a especificação, descrita através de uma linguagem fluente e carregada a partir de um ficheiro, da interface gráfica de aplicações baseadas no *framework* `System.Windows.Forms`.

O sistema segue o princípio da separação entre *view* e *controller*, segundo o padrão MVC. No documento é especificada a *view* e indicada a classe, como atributo do elemento raíz, que se constitui como *controller* da aplicação.

O mapeamento entre os eventos gerados pelos controlos e os respectivos *handlers*, presentes na classe *controller*, é feito por convenção de nomes. A seguir apresenta-se a gramática `BNF` que especifica a convenção seguida:

```
<Nome de handler> ::=  [<contexto>'_'][<elementos>'_']<evento>
```

O nome de um *handler* pode ter até três palavras, separadas pelo caracter *underscore*('_'). As duas primeiras palavras (`<contexto>` e `<elementos>`) são opcionais e indicam o nome de um tipo de controlo ou de um controlo específico. A primeira palavra indica a(s) zona(s) do *form* a que o *handler* se aplica . A segunda palavra indica o(s) controlo(s), dentro dessas zonas, a que o *handler* se aplica. Quando a primeira ou segunda palavras corresponderem ao nome de um tipo, consideram-se os controlos desse tipo ou de tipos derivados. Finalmente, a terceira palavra indica o evento a que o *handler* se aplica. Por exemplo, se o nome for constituído por uma só palavra, trata-se de um *handler* do evento indicado no nome, que será aplicado a todos os controlos do *form*.

Não deve ser tentado o registo de um método com uma assinatura incompatível com o evento especificado no seu nome.

Primeiro exemplo

Na figura 1 é apresentado o documento que especifica a *view* (form) de uma aplicação gráfica de exemplo.

```
using System;
using System.Windows.Forms;
using ChelasUIMaker.Engine;
namespace ChelasUIMakerApp{
  internal class MyConfig : Config{

   protected override void LoadConfig(){
    var area1 = DefineArea<Panel>(p => { p.Width = 390; })
                .WithContent<TextBox>(
                      DefineArea<TextBox>(tb => { tb.Name = "visor"; tb.Dock = DockStyle.Fill; }))
                .WithContent<Button>(DefineArea<Button>(b1 => { b1.Top = 50; b1.Text = "Botao 1"; }))
                .WithContent<Button>(DefineArea<Button>(b2 => { b2.Top = 75; b2.Text = "Botao 2"; }));

    var area2 = DefineArea<Panel>(p => { p.Top = 100; p.Width = 390; })
                .WithContent<Button>(DefineArea<Button>(b1 => { b1.Top = 50; b1.Text = "Botao 3"; }))
                .WithContent<Button>(DefineArea<Button>(
                          b2 => { b2.Top = 50; b2.Left = 300; b2.Name = "B4"; b2.Text = "Botao 2"; }));

    DefineArea<Form>(f => { f.Text = "Form especificado de forma declarativa"; f.Width = 430; })
         .WithController<MyController>()
         .WithContent(area1)
         .WithContent(area2);
        }
    }
}
```

Fig. 1 Descrição em linguagem fluente do *Form* do primeiro exemplo

Note-se que as *tags* correspondem a controlos do *namespace* `System.Windows.Forms`, presentes no *assembly* `System.Windows.Forms,version=4.0.0.0,Culture=neutral,PublicKeyToken=b77a5c561934e089`. O nome dos atributos corresponde a propriedades presentes no respectivo controlo, à excepção do atributo `controller` do elemento `Form`, que indica o *assembly* e o tipo desse *assembly* que se constitui como *controller* da *view* especificada. A infraestrutura terá de saber converter os valores dos atributos (*strings*) em valores dos tipos das propriedades. No mínimo, serão suportadas propriedades de tipos primitivos, *strings* e tipos enumerados.

Na figura 2 é apresentado o *controller* correspondente. O exemplo mostra que os *controllers* derivam da classe `Controller`, implementada na infraestrutura, e que fornece alguns serviços básicos, nomeadamente a propriedade (`View`) que refere a *view* a que o *controller* se aplica. A *view* tem um *indexer* que permite ao *controller* aceder por nome aos respectivos controlos.

Note-se que o nome dos *handlers* apenas indicam o nome do evento, pelo que serão registados como *listeners* do evento `MouseEnter` e `MouseLeave` em todos os controlos do *form*. Neste caso, o mesmo efeito seria conseguido, para o evento `MouseEnter`, com o nomes `Control_MouseEnter` ou `Form_Control_MouseEnter`. Caso o nome fosse `Panel_Control_MouseEnter` o *handler* apenas seria registado nos controlos filhos (directos e indirectos) de controlos do tipo `Panel`.

```
using System;
using System.Windows.Forms;
using ChelasUIMaker.Engine;
namespace ChelasUIMakerApp{
   public class MyController : Controller {

          protected void MouseEnter(object sender, EventArgs args) {
                Control c = (Control)sender;
                View["visor"].Text = sender.GetType().FullName + "(" + c.Name + ")";
          }

          protected void MouseLeave(object sender, EventArgs args) {
                Control c = (Control)sender;
                View["visor"].Text = "";
          }
   }
}
```

Fig. 2- *Controller* do primeiro exemplo

Finalmente é apresentado na figura 3 um *snapshot* da execução da aplicação de exemplo. Repare que o cursor está a passar pelo botão de texto `'Botão 4'`, pelo que na *textbox* de nome `visor` é apresentado o tipo e o nome do respectivo controlo. Na figura 4 apresenta-se a possível implementação do código do motor que lança a aplicação.

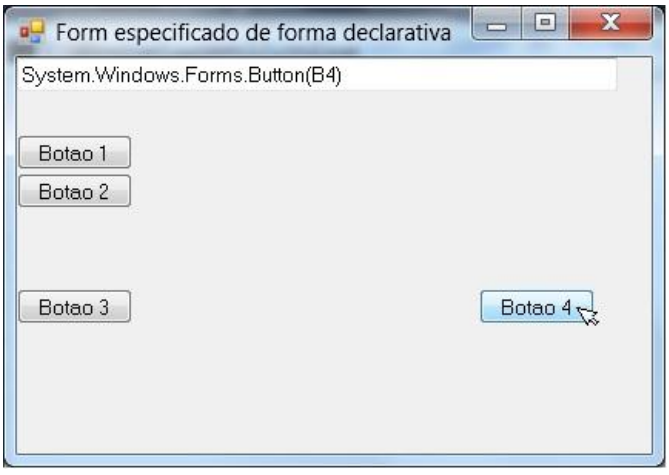

Fig. 3 Execução da aplicação de exemplo

```
using System;
using ChelasUIMaker.Engine;
using System.Windows.Forms;

namespace ChelasUIMakerApp
{


    class Program
    {
        static void Main(string[] args)
        {

            XView xv = ViewEngine.LoadConfig(new MyConfig());
            Application.Run((Form)xv.Control);
        }
    }
}
```

Fig. 4 Possível ponto de entrada da infraestrutura

Segundo exemplo

A seguir apresenta-se um segundo exemplo com a especificação de uma calculadora básica, com aritmética inteira. O código do *controller* é **fornecido em código em anexo**. Na figura 5 é apresentado um *snapshot* da execução da aplicação. Na figura 6 apresenta-se a correspondente especificação.

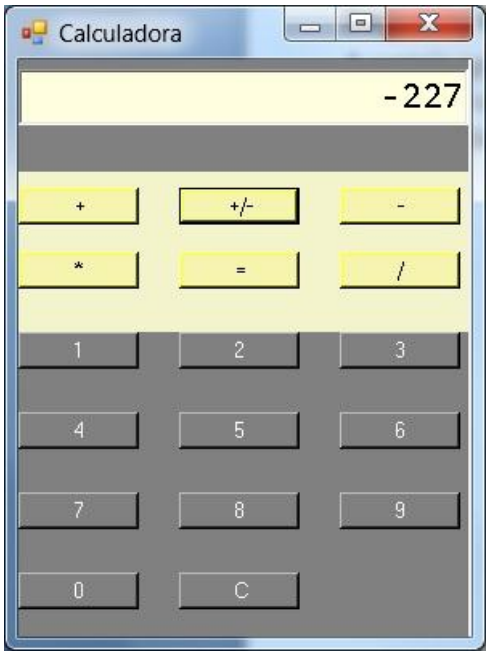

Fig. 5 – Snapshot da aplicação calculadora

```
using System;
using System.Windows.Forms;
using System.Drawing;
using ChelasUIMaker.Engine;


namespace ChelasUIMakerCalcApp{
 internal class MyCalcConfig : Config{

   protected override void LoadConfig(){

     var area1 = DefineArea<Panel>(p => { p.Height=100; p.Width = 280; p.BackColor=Color.Gray; })
                .WithContent<TextBox>(DefineArea<TextBox>(tb => { tb.Top = 6; tb.Enabled=false;
                                    tb.Name="visor";
                                    tb.BackColor=Color.LightYellow; tb.Text="0";
                                    tb.TextAlign=HorizontalAlignment.Right;
                                    tb.SetFont("Consolas,14"); }));


    var area2 = DefineArea<Panel>(p => { p.Name="operations"; p.Top=70; p.Width=280; p.Height=100;
                                         p.SetBackColor(0x30FFFF30); })
                .WithContent<Button>(DefineArea<Button>(b1 => { b1.Text="+"; b1.Top = 10;}))
              .WithContent<Button>(DefineArea<Button>(b2 => { b2.Text="+/-"; b2.Top = 10; b2.Left=100;}))
                .WithContent<Button>(DefineArea<Button>(b3 => { b3.Text="-"; b3.Top = 10; b3.Left=200;}))
                .WithContent<Button>(DefineArea<Button>(b4 => { b4.Text="*"; b4.Top = 50;}))
                .WithContent<Button>(DefineArea<Button>(b5 => { b5.Text="/"; b5.Top = 50; b5.Left=200;}))
                .WithContent<Button>(DefineArea<Button>(b6 => { b6.Text="="; b6.Top = 50;b6.Left=100;}));


   var area3 = DefineArea<Panel>(p => { p.Top = 170; p.Width = 280; p.Height=200;
                                        p.BackColor=Color.Gray; })
               .WithContent<Button>(
                   DefineArea<Button>(b1 => { b1.Text = "1"; b1.ForeColor = Color.White; }))
               .WithContent<Button>(
                   DefineArea<Button>(b2 => { b2.Text = "2"; b2.Left=100; b2.ForeColor = Color.White; }))
               .WithContent<Button>(
                   DefineArea<Button>(b3 => { b3.Text = "3"; b3.Left=200; b3.ForeColor = Color.White; }))
               .WithContent<Button>(
                   DefineArea<Button>(b4 => { b4.Text = "4"; b4.Top=50; b4.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           b5 => { b5.Text = "5";  b5.Top=50; b5.Left=100; b5.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           b6 => { b6.Text = "6"; b6.Top=50; b6.Left=100; b6.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           b7 => { b7.Text = "7"; b7.Top=100; b7.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           b8 => { b8.Text = "8"; b8.Top=100; b8.Left=100; b8.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           b9 => { b9.Text = "9"; b9.Top=100; b9.Left=200; b9.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           b0 => { b0.Text = "0"; b0.Top=150; b0.ForeColor = Color.White; }))
               .WithContent<Button>(DefineArea<Button>(
                           bC => { bC.Text = "C"; bC.Top=150; bC.Left=200; bC.ForeColor =Color.White; }));


    DefineArea<Form>(f => { f.Text = "Calculadora"; f.Height = 400;f.Width=300; })
        .WithController<CalcController>()
        .WithContent(area1)
        .WithContent(area2)
        .WithContent(area3);
       }
  }
 }
```

Fig. 6 - Especificação da *view* da calculadora

Características Opcionais

Suporte a propriedades com tipos complexos

Repare que no exemplo da calculadora são usadas propriedades de tipos complexos, nomeadamente a propriedade `Size`, a propriedade `Font` e a propriedade `BackColor`. A ser implementado, o suporte a propriedades de tipos complexos deverá ser extensível, isto é, permitir o acrescento de novos tipos de propriedades sem alterar o motor da infraestrutura.

Gestores de layouts

Outro aspecto que pode ser melhorado prende-se com a utilização de posicionamento absoluto (relativo ao *container*), na colocação dos controlos. Seria útil poder integrar gestores de layout, por exemplo um *grid layout* para automatizar o posicionamento dos botões da calculadora.

Prazo de Entrega

O trabalho terá de ser entregue até ao dia **1 de Setembro de 2011**.