Aula 6 - Padrões de Projeto Adicionais

Docupedia Export

Author:Goncalves Donathan (CtP/ETS)
Date:23-May-2023 14:02

Table of Contents

1 Bridge

O Bridge é um padrão estrutural que é estrutualmente semelhane a padrões como o Strategy, porém, com uma finalidade e uso totalmente diferente. Sua ideia é reduzir o trabalho e diminuir os possíveis erros e inconsistências ao modificar um sistema. A ideia dele é separar implementação de abstração. A ideia é bem complexa, na verdade, e dificil de compreender muitas vezes, mas observe o exemplo a seguir.

```
1  using System;
2  using System.Text;
3  using System.Collections.Generic;
4
5  List<Product> products = new List<Product>()
6  {
7      new Product()
8      {
9          Name = "Xispita",
10         Price = 10.5f
11     },
12     new Product()
13     {
14         Name = "Caixa de Ponteiros",
15         Price = 3f
16     }
17 };
18
19 Abstraction abstraction = new Abstraction();
20
21 abstraction.implementation = new NormalImplementation();
22 abstraction.DrawMarket("Treviloja normal", products);
23
24 Console.WriteLine();
25 Console.WriteLine();
26 Console.WriteLine();
27
28 abstraction.implementation = new SpecialImplementation();
29 abstraction.DrawMarket("Treviloja especial", products);
30
31 public class Product
32 {
33     public string Name { get; set; }
```

```
34      public float Price { get; set; }
35  }
36
37  public class Abstraction
38  {
39      public Implementation implementation { get; set; }
40
41      public virtual void DrawMarket(string title, List<Product> products)
42      {
43          implementation.AddString(title);
44          implementation.AddNewLine();
45
46          foreach (var product in products)
47          {
48              implementation.AddString(product.Name);
49              implementation.AddFloat(product.Price);
50              implementation.AddNewLine();
51          }
52
53          implementation.Print();
54      }
55  }
56
57  public interface Implementation
58  {
59      void AddString(string s);
60      void AddFloat(float f);
61      void AddNewLine();
62      void Print();
63  }
64
65  public class NormalImplementation : Implementation
66  {
67      StringBuilder sb = new StringBuilder();
68
69      public void AddFloat(float s)
70          => sb.Append(s);
71
72      public void AddNewLine()
```

```
73          => sb.AppendLine();
74
75      public void AddString(string f)
76          => sb.Append(f);
77
78      public void Print()
79      {
80          Console.Write(sb.ToString());
81          sb.Clear();
82      }
83  }
84
85  public class SpecialImplementation : Implementation
86  {
87      int size = 0;
88      List<string> lines = new List<string>();
89      StringBuilder sb = new StringBuilder();
90
91      public void AddString(string s)
92      {
93          size += s.Length;
94          sb.Append(s);
95      }
96
97      public void AddFloat(float f)
98      {
99          string str = $"{f: 0.00}";
100         AddString(str);
101     }
102
103     public void Print()
104     {
105         foreach (var line in lines)
106             Console.WriteLine(line);
107         lines.Clear();
108     }
109
110     public void AddNewLine()
111     {
```

```
112             formatString();
113             lines.Add("");
114         }
115 
116         private void formatString()
117         {
118             sb.Insert(0, "\n");
119             sb.AppendLine();
120             for (int i = 0; i < size; i++)
121             {
122                 sb.Insert(0, "-");
123                 sb.Append("-");
124             }
125             lines.Add(sb.ToString());
126             sb.Clear();
127             size = 0;
128         }
129     }
```

O exemplo acima é muito interessante. A abstração usa as funções da implementação para construir desenhar uma loja com produtos. A depender da implementação, a forma com que a loja será desenhada muda. Isso é perfeito, pois podemos mudar a loja inteira sem alterar a abstração e sem alterar as implementações existentes, basta criar uma nova implementação. Além disso, se modificarmos a abstração ou criarmos classes filhas da abstração que fazem mais coisas, reescrevendo métodos ou adicionando, não quebramos o funcionamento da implementação que fica separada a parte.

2 Command

Command é um padrão comportamental usado para encapsular e parametrizar comandos que podem ser ativados em vários momentos. Geralmente o Command não é utilizado em linguagens que possuem programação funcional, como C#, mas ainda sim ele possui várias utilizações poderosas.Ele é útil quando queremos permitir alta configuração na funcionalidade dos comandos. Abaixo um simples editor de texto com a possibilidade de gravar macros usando command.

```
1   using System;
2   using System.Linq;
3   using System.Collections.Generic;
4
5   Invoker app = new Invoker();
6
7   while (true)
8   {
9       Console.WriteLine(app.Text);
10      string command = Console.ReadLine();
11      app.UseCommmand(command);
12      Console.Clear();
13  }
14
15  public class Invoker
16  {
17      public string Text { get; set; }
18
19      public Invoker()
20      {
21          commmandDict.Add("add", new AddCommand());
22          commmandDict.Add("rev", new ReverseCommand());
23          commmandDict.Add("sub", new SubStringCommand());
24      }
25
26      public IEnumerable<string> Commands => commmandDict.Keys.Append("macro");
27
28      private Dictionary<string, ICommand> commmandDict = new Dictionary<string, ICommand>();
29      private Macro macro = null;
30      private bool macroMode = false;
31
32      public void UseCommmand(string comm)
33      {
```

```
34          comm = comm.Trim();
35          var parts = comm.Split(' ', StringSplitOptions.RemoveEmptyEntries);
36          if (parts.Length == 0)
37              return;
38
39          if (parts[0] == "help")
40          {
41              Console.WriteLine("Comandos Disponíveis:");
42              foreach (var com in Commands)
43                  Console.WriteLine(com);
44              Console.ReadKey(true);
45              return;
46          }
47
48          if (parts[0] == "macro")
49          {
50              macroMode = !macroMode;
51              if (macroMode)
52                  macro = new Macro(parts[1]);
53              else
54                  commmandDict.Add(macro.Name, macro);
55              return;
56          }
57
58          if (macroMode)
59          {
60              macro.Add(commmandDict[parts[0]], parts.Skip(1).ToArray());
61              return;
62          }
63
64          if (!this.commmandDict.ContainsKey(parts[0]))
65          {
66              Console.WriteLine("Commando não existe!");
67              Console.ReadKey(true);
68              return;
69          }
70
71          this.Text = commmandDict[parts[0]].Apply(this.Text, parts.Skip(1).ToArray());
72      }
```

```
73  }
74
75  public interface ICommand
76  {
77      string Apply(string text, string[] parameters);
78  }
79
80  public class ReverseCommand : ICommand
81  {
82      public string Apply(string text, string[] parameters)
83          => string.Concat(text.Reverse());
84  }
85
86  public class SubStringCommand : ICommand
87  {
88      public string Apply(string text, string[] parameters)
89      {
90          int size = int.Parse(parameters[0]);
91          int sizeoff = int.Parse(parameters[1]);
92
93          if (sizeoff > text.Length)
94              return string.Empty;
95
96          if (sizeoff + size >= text.Length)
97              return text.Substring(sizeoff);
98
99          return text.Substring(sizeoff, size);
100     }
101 }
102
103 public class AddCommand : ICommand
104 {
105     public string Apply(string text, string[] parameters)
106     {
107         string addedText = parameters.Aggregate("", (a, s) => a + s + " ");
108         addedText = addedText.Substring(0, addedText.Length - 1);
109         return text + addedText;
110     }
111 }
```

```
112
113 public class Macro : ICommand
114 {
115     private List<ICommand> commands = new List<ICommand>();
116     private List<string[]> parameters = new List<string[]>();
117
118     public string Name { get; set; }
119     public Macro(string name)
120         => this.Name = name;
121
122     public void Add(ICommand command, string[] parameter)
123     {
124         this.commands.Add(command);
125         this.parameters.Add(parameter);
126     }
127
128     public string Apply(string text, string[] parameters)
129     {
130         for (int i = 0; i < this.parameters.Count; i++)
131             text = commands[i].Apply(text, this.parameters[i]);
132         Console.WriteLine(text);
133         return text;
134     }
135 }
```

3 Memento

Memento é padrão comportamental com o objetivo de criar sistemas com operações de salvar/restaurar:

```
using System;
using System.Text;
using System.Collections.Generic;

State state = new State();
Caretaker history = new Caretaker();

for (int i = 0; i < 8; i++)
    state.Paint(i, 0, 4);

var mem = new Memento(state);
history.Save(mem);
Console.WriteLine(state);

for (int i = 0; i < 8; i++)
    state.Paint(i, 1, 3);

mem = new Memento(state);
history.Save(mem);
Console.WriteLine(state);

for (int i = 0; i < 8; i++)
    state.Paint(i, 2, 2);

mem = new Memento(state);
history.Save(mem);
Console.WriteLine(state);

mem = history.Undo();
state = mem.GetState();
Console.WriteLine(state);

mem = history.Undo();
state = mem.GetState();
Console.WriteLine(state);
```

```
36
37  public class Caretaker
38  {
39      Stack<Memento> stack = new Stack<Memento>();
40
41      public void Save(Memento memento)
42          => stack.Push(memento);
43
44      public Memento Undo()
45      {
46          stack.Pop();
47          return stack.Peek();
48      }
49  }
50
51  public class Memento
52  {
53      private byte[] state;
54
55      public Memento(State state)
56      {
57          var arr = state.Data;
58          var copy = new byte[arr.Length];
59          arr.CopyTo(copy, 0);
60          this.state = copy;
61      }
62
63      public State GetState()
64          => new State(this.state);
65  }
66
67  public class State
68  {
69      public byte[] Data => data;
70      private byte[] data = new byte[64];
71
72      public State() { }
73
74      public State(byte[] data)
```

```
        => this.data = data;

    public void Paint(int i, int j, byte value)
    {
        this.data[i + 8 * j] = value;
    }

    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();

        for (int j = 0; j < 8; j++)
        {
            for (int i = 0; i < 8; i++)
            {
                var value = data[8 * j + i];

                if (value == 0)
                    sb.Append(" ");
                else if (value == 1)
                    sb.Append("░");
                else if (value == 2)
                    sb.Append("▒");
                else if (value == 3)
                    sb.Append("▓");
                else if (value > 3)
                    sb.Append("█");
            }
            sb.AppendLine();
        }

        return sb.ToString();
    }
}
```

4 Prototype

As vezes queremos criar cópias de objetos mas não sabemos exatamente como eles são. Objetos podem ter campos escondidos que desconhecemos. Por exemplo, a classe Random que é bem mais complexa por dentro do que por fora. Para isso, podemos usar o padrão prototype onde definimos um método de clonagem dentro do próprio objeto.

```
1   BoxPrototype box = new BoxPrototype("Olá, Mundo!\nOlá, Mundo!", 2);
2   box.Open();
3
4   var ohter = box.Clone() as BoxPrototype;
5   ohter.Add();
6   ohter.Open();
7
8   box.Open();
9
10  public interface IPrototype
11  {
12      IPrototype Clone();
13  }
14
15  public class BoxPrototype : IPrototype
16  {
17      private string content;
18      private int quantity;
19
20      public BoxPrototype(string content, int quantity)
21      {
22          this.content = content;
23          this.quantity = quantity;
24      }
25
26      public void Open()
27      {
28          for (int i = 0; i < quantity; i++)
29              Console.WriteLine(content);
30      }
31
32      public void Add()
33          => quantity++;
```

```
34
35      public IPrototype Clone()
36      {
37          BoxPrototype copy = new BoxPrototype(this.content, this.quantity);
38          return copy;
39      }
40  }
```

Assim podemos clonar a caixa mesmo sem ter certeza de seu conteúdo. Existe uma interface no C# para esse propósito chamada IClonable.

5 Exemplo: Draw.io multiplataforma

setup.ps1

```
mkdir DrawIoDesktop
mkdir DrawIoWeb
mkdir DrawIoLib

cd DrawIoLib
dotnet new classlib
cd ..

cd DrawIoDesktop
dotnet new winforms
dotnet add reference ..\DrawIoLib\DrawIoLib.csproj
cd ..

cd DrawIoWeb
dotnet new blazorserver
dotnet add reference ..\DrawIoLib\DrawIoLib.csproj
dotnet add package Blazor.Extensions.Canvas
cd ..
```

IVisualBehavior.cs

```
using System.Drawing;
using System.Threading.Tasks;

namespace DrawIo;

public interface IVisualBehavior
{
    Task FillRectangle(RectangleF rect, Color color);
    Task DrawRectangle(RectangleF rect, Color color);
    Task DrawText(RectangleF rect, string text);
    Task DrawLine(PointF p, PointF q, Color color, float width);
```

```
12      Task DrawDottedLine(PointF p, PointF q, Color color, float width);
13  }
```

IPrototype.cs

```
1   namespace DrawIo;
2
3   public interface IPrototype
4   {
5       VisualObject Clone();
6   }
```

VisualObject.cs

```
1   using System;
2   using System.Threading.Tasks;
3
4   namespace DrawIo;
5
6   public abstract class VisualObject : ICloneable, IPrototype
7   {
8       protected IVisualBehavior g;
9
10      public VisualObject(IVisualBehavior g)
11          => this.g = g;
12
13      public abstract Task Draw(bool selected);
14
15      public abstract VisualObject Clone();
16
17      object ICloneable.Clone()
18          => this.Clone();
19  }
```

ClassBox

```
using System.Drawing;
using System.Threading.Tasks;

namespace DrawIo;

public class ClassBox : VisualObject
{
    public RectangleF Rectangle { get; set; }
    public Color Color { get; set; }
    private string text;

    public ClassBox(IVisualBehavior g, PointF initial) : base(g)
    {
        this.text = "Classe";
        this.Color = Color.White;
        this.Rectangle = new RectangleF(initial, new SizeF(100, 100));
    }

    public override VisualObject Clone()
    {
        var crrPt = this.Rectangle.Location;
        var newPt = new PointF(crrPt.X + 20, crrPt.Y + 20);

        ClassBox box = new ClassBox(this.g, PointF.Empty);
        box.text = this.text;
        box.Color = this.Color;
        box.Rectangle = new RectangleF(newPt, this.Rectangle.Size);

        return box;
    }

    public override async Task Draw(bool selected)
    {
        await g.FillRectangle(Rectangle, Color);
        await g.DrawRectangle(Rectangle, selected ? Color.Red : Color.Black);
```

```
36          await g.DrawText(Rectangle, text);
37      }
38  }
```

Arrow.cs

```
1   using System.Drawing;
2   using System.Threading.Tasks;
3
4   namespace DrawIo;
5
6   public class Arrow : VisualObject
7   {
8       private bool dotted = false;
9       private ClassBox start;
10      private ClassBox end;
11
12      public Arrow(IVisualBehavior g, ClassBox start, ClassBox end, bool dotted) : base(g)
13      {
14          this.start = start;
15          this.end = end;
16          this.dotted = dotted;
17      }
18
19      public override VisualObject Clone()
20      {
21          Arrow arrow = new Arrow(g, start, end, dotted);
22          return arrow;
23      }
24
25      public override async Task Draw(bool selected)
26      {
27          var left = start.Rectangle.Location.X > end.Rectangle.Location.X
28              ? end : start;
29          var right = left == start ? end : start;
30
31          var leftPt = new PointF(
32              left.Rectangle.Location.X + left.Rectangle.Width,
```

```
                left.Rectangle.Location.Y + left.Rectangle.Height / 2
            );
            var rightPt = new PointF(
                right.Rectangle.Location.X,
                right.Rectangle.Location.Y + right.Rectangle.Height / 2
            );

            float wid = rightPt.X - leftPt.X;
            float middle = leftPt.X + wid / 2;
            var middleLeftPt = new PointF(middle, leftPt.Y);
            var middleRightPt = new PointF(middle, rightPt.Y);

            var color = selected ? Color.Red : Color.Black;

            if (dotted)
            {
                await g.DrawDottedLine(leftPt, middleLeftPt, color, 2f);
                await g.DrawDottedLine(middleLeftPt, middleRightPt, color, 2f);
                await g.DrawDottedLine(middleRightPt, rightPt, color, 2f);
            }
            else
            {
                await g.DrawLine(leftPt, middleLeftPt, color, 2f);
                await g.DrawLine(middleLeftPt, middleRightPt, color, 2f);
                await g.DrawLine(middleRightPt, rightPt, color, 2f);
            }

            if (end == left)
            {
                await g.DrawLine(leftPt, new PointF(leftPt.X + 20, leftPt.Y + 10), color, 3f);
                await g.DrawLine(leftPt, new PointF(leftPt.X + 20, leftPt.Y - 10), color, 3f);
            }
            else
            {
                await g.DrawLine(rightPt, new PointF(rightPt.X - 20, rightPt.Y + 10), color, 3f);
                await g.DrawLine(rightPt, new PointF(rightPt.X - 20, rightPt.Y - 10), color, 3f);
            }
        }
```

```
71  }
```

ICommand.cs

```
1   namespace DrawIo;
2
3   public interface ICommand
4   {
5       void Execute(Project app);
6       void Undo(Project app);
7   }
```

Project.cs

```
1   using System.Collections.Generic;
2   using System.Threading.Tasks;
3
4   namespace DrawIo;
5
6   public class Project
7   {
8       public VisualObject Selected { get; set; }
9       public IEnumerable<VisualObject> Objects => this.objs;
10      private VisualObject copied = null;
11      private List<VisualObject> objs = new List<VisualObject>();
12      private Stack<ICommand> history = new Stack<ICommand>();
13      private Stack<ICommand> undone = new Stack<ICommand>();
14
15      public Project()
16      {
17
18      }
19
20      public void Execute(ICommand command)
21      {
22          command.Execute(this);
```

```
23          history.Push(command);
24          undone.Clear();
25      }
26
27      public void Undo()
28      {
29          if (history.Count < 1)
30              return;
31
32          var commmand = history.Pop();
33          commmand.Undo(this);
34          this.undone.Push(commmand);
35      }
36
37      public void Copy()
38          => copied = Selected;
39
40      public void Cut()
41      {
42          copied = Selected;
43          Remove(Selected);
44          Selected = null;
45      }
46
47      public VisualObject Paste()
48      {
49          if (copied == null)
50              return null;
51
52          var copy = copied.Clone();
53          this.objs.Add(copy);
54          copied = copy;
55          return copy;
56      }
57
58      public VisualObject Delete()
59      {
60          Remove(Selected);
61          var removed = Selected;
```

```
62          Selected = null;
63          return removed;
64      }
65
66      public void Remove(VisualObject obj)
67          => this.objs.Remove(obj);
68
69      public void Add(VisualObject obj)
70          => this.objs.Add(obj);
71
72      public void Redo()
73      {
74          if (undone.Count < 1)
75              return;
76
77          var commmand = undone.Pop();
78          commmand.Execute(this);
79          this.history.Push(commmand);
80      }
81
82      public async Task Draw()
83      {
84          foreach (var obj in this.objs)
85          {
86              await obj.Draw(obj == Selected);
87          }
88      }
89  }
```

AddCommand.cs

```
1   namespace DrawIo.Commands;
2
3   public class AddCommand : ICommand
4   {
5       public VisualObject Object { get; set; }
6       public void Execute(Project app)
7       {
```

```
            app.Add(Object);
        }

        public void Undo(Project app)
        {
            app.Remove(Object);
        }
    }
```

DeleteCommand.cs

```
namespace DrawIo.Commands;

public class DeleteCommand : ICommand
{
    private VisualObject deleted = null;

    public void Execute(Project app)
    {
        this.deleted = app.Delete();
    }

    public void Undo(Project app)
    {
        if (this.deleted == null)
            return;

        app.Add(this.deleted);
    }
}
```

MoveCommand.cs

```
using System.Drawing;

namespace DrawIo.Commands;
```

```
4
5   public class MoveCommand : ICommand
6   {
7       public ClassBox Object { get; set; }
8       public PointF Old { get; set; }
9       public PointF New { get; set; }
10
11      public void Execute(Project app)
12      {
13          this.Old = Object.Rectangle.Location;
14          Object.Rectangle = new RectangleF(New, this.Object.Rectangle.Size);
15      }
16
17      public void Undo(Project app)
18      {
19          Object.Rectangle = new RectangleF(Old, this.Object.Rectangle.Size);
20      }
21  }
```

PasteCommand.cs

```
namespace DrawIo.Commands;

public class PasteCommand : ICommand
{
    private VisualObject pasted = null;

    public void Execute(Project app)
    {
        this.pasted = app.Paste();
    }

    public void Undo(Project app)
    {
        if (this.pasted == null)
            return;
```

```
            app.Remove(this.pasted);
        }
    }
```

WebVisualBehavior.cs

```
 1  using System.Drawing;
 2
 3  using DrawIo;
 4  using Blazor.Extensions.Canvas;
 5  using Blazor.Extensions.Canvas.Canvas2D;
 6
 7  public class WebVisualBehavior : IVisualBehavior
 8  {
 9      private Canvas2DContext context = null;
10
11      public WebVisualBehavior(Canvas2DContext context)
12          => this.context = context;
13
14      public Task DrawDottedLine(PointF p, PointF q, Color color, float width)
15      {
16          throw new NotImplementedException();
17      }
18
19      public async Task DrawLine(PointF p, PointF q, Color color, float width)
20      {
21          await context.SetFillStyleAsync(colorToString(color));
22          await context.BeginPathAsync();
23          await context.MoveToAsync(p.X, p.Y);
24          await context.LineToAsync(q.X, q.Y);
25          await context.StrokeAsync();
26      }
27
28      public async Task DrawRectangle(RectangleF rect, Color color)
29      {
30          await context.BeginPathAsync();
31          await context.SetFillStyleAsync(colorToString(color));
32          await context.RectAsync(rect.X, rect.Y, rect.Width, rect.Height);
```

```
33          await context.StrokeAsync();
34      }
35
36      public async Task DrawText(RectangleF rect, string text)
37      {
38          await context.SetFontAsync("20px Calibri");
39          await context.SetStrokeStyleAsync(colorToString(Color.Black));
40          await context.StrokeTextAsync(text, rect.X, rect.Y + rect.Height / 2);
41      }
42
43      public async Task FillRectangle(RectangleF rect, Color color)
44      {
45          await context.SetFillStyleAsync(colorToString(color));
46          await context.FillRectAsync(rect.X, rect.Y, rect.Width, rect.Height);
47      }
48
49      private string colorToString(Color color)
50          => $"rgb({color.R}, {color.G}, {color.B})";
51  }
```

_Layout.cshtml

```
1   @using Microsoft.AspNetCore.Components.Web
2   @namespace DrawIoWeb.Pages
3   @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
4
5   <!DOCTYPE html>
6   <html lang="en">
7   <head>
8       <meta charset="utf-8" />
9       <meta name="viewport" content="width=device-width, initial-scale=1.0" />
10      <base href="~/" />
11      <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
12      <link href="css/site.css" rel="stylesheet" />
13      <link href="DrawIoWeb.styles.css" rel="stylesheet" />
14      <component type="typeof(HeadOutlet)" render-mode="ServerPrerendered" />
15      <script src="_content/Blazor.Extensions.Canvas/blazor.extensions.canvas.js"></script>
16  </head>
```

```
<body>
    @RenderBody()

    <div id="blazor-error-ui">
        <environment include="Staging,Production">
            An error has occurred. This application may no longer respond until reloaded.
        </environment>
        <environment include="Development">
            An unhandled exception has occurred. See browser dev tools for details.
        </environment>
        <a href="" class="reload">Reload</a>
        <a class="dismiss">✕</a>
    </div>

    <script src="_framework/blazor.server.js"></script>
</body>
</html>
```

MainLayout.razor

```
@inherits LayoutComponentBase

<PageTitle>DrawIoWeb</PageTitle>

<div class="page">
    @Body
</div>
```

Index.razor

```
@page "/"
@using DrawIo;
@using System.Drawing;
@using Blazor.Extensions;
@using Blazor.Extensions.Canvas
@using Blazor.Extensions.Canvas.Canvas2D;
```

```
7
8   <BECanvas Width="600" Height="400" @ref="_canvasReference"></BECanvas>
9
10  <br/>
11  <button @onclick="Adicionar">Adicionar</button>
12
13  @code
14  {
15      private Canvas2DContext _context;
16      protected BECanvasComponent _canvasReference;
17      private WebVisualBehavior vb;
18
19      private Project prj;
20      int elements = 0;
21
22      protected override async Task OnAfterRenderAsync(bool firstRender)
23      {
24          this._context = await this._canvasReference.CreateCanvas2DAsync();
25          vb = new WebVisualBehavior(_context);
26
27          prj = new Project();
28          ClassBox b1 = new ClassBox(vb, new PointF(50, 50));
29          ClassBox b2 = new ClassBox(vb, new PointF(300, 300));
30
31          prj.Add(b1);
32          prj.Add(b2);
33          prj.Add(new Arrow(vb, b1, b2, false));
34
35          await prj.Draw();
36      }
37
38      private async void Adicionar()
39      {
40          elements++;
41          ClassBox box = new ClassBox(vb, new PointF(50 + 20 * elements, 50 + 20 * elements));
42          prj.Add(box);
43          prj.Selected = box;
44
45          await prj.Draw();
```

```
46          }
47      }
```

6 Exercícios

Faça um simples editor de texto que possa rodar em multiplas plataformas e implemente na plataforma Console. Use Bridge para reduzir o trabalho a ser feito. Use Memento para permitir que o texto possa ter alterações desfeitas e refeitas.