Aula 5 - Padrões de Projeto Comportamentais

Docupedia Export

Author:Belizario Marcos (CtP/ETS)
Date:19-May-2023 12:53

Table of Contents

- Proxy
- Padrões de Projeto Comportamentais
- Template Method
- Strategy
- State
- Exemplo: Bot para jogos digitais
- Exercícios

1 Proxy

Proxy é um Padrão Estrutural para controlar o acesso a recursos que precisam de tempo para serem processados. Ele pode te ajudar a acessar um banco de dados, buscar imagens na Web e mostrar uma imagem mais simples enquanto a mesma carrega, mostrar um calculo simples enquanto um calculo complexo carrega ou carregar dados em Cache. Todas essas operações são feitas com o padrão Proxy:

```
1  using System;
2  using System.Threading;
3
4  Service service = new Service();
5  Proxy proxy = new Proxy(service);
6
7  byte[] arr = new byte[1024 * 1024 * 1024];
8  Random rand = new Random();
9  rand.NextBytes(arr);
10
11 Console.Clear();
12 while (true)
13 {
14     Console.SetCursorPosition(0, 0);
15     Console.WriteLine(proxy.GetSum(arr));
16 }
17
18
19 public interface IService
20 {
21     string GetSum(byte[] arr);
22 }
23
24 public class Service : IService
25 {
26     public string GetSum(byte[] arr)
27     {
28         long sum = 0;
29         for (int i = 0; i < arr.Length; i++)
30             sum += arr[i];
31         return $"Total: {sum} (exato)";
32     }
33 }
```

```
34
35  public class Proxy : IService
36  {
37      private IService realService = null;
38      private string realOutput = null;
39      private Thread thread = null;
40
41      public Proxy(IService service)
42          => this.realService = service;
43
44      public string GetSum(byte[] arr)
45      {
46          if (realOutput != null)
47              return realOutput;
48
49          if (thread == null)
50              startService(arr);
51
52          return $"Total: {(128 * arr.LongLength)} (aprox)";
53      }
54
55      private void startService(byte[] arr)
56      {
57          thread = new Thread(() =>
58          {
59              var result = realService.GetSum(arr);
60              this.realOutput = result;
61          });
62          thread.Start();
63      }
64  }
```

Neste exemplo, apresentamos uma mensagem diferente enquanto esperamos o resultado concreto.
Note que o Proxy lembra muito o Decorator. A diferença fundamental entre eles é o proposito. Muitos padrões são estruturalmente parecidos e o que os diferencia é como você os aplica para resolver os problemas e quais tipos de problemas você quer resolver.

2 Padrões de Projeto Comportamentais

Nosso último grupo de padrões de projeto tem por objetivo representar comportamentos. Antes criamos objetos complexos, depois construímos estruturas complexas, agora vamos representar comportamentos complexos.

3 Template Method

Nosso primeiro padrão é outro muito simples de se compreender e usar. O Template Method permite que você faça a implementação parcial de alguma funcionalidade nas classes bases.

```
1  using System;
2
3  BaseClass a = new ImplA();
4  BaseClass b = new ImplB();
5
6  a.Func(); // Hello, World!
7  b.Func(); // Olá... Mundo?
8
9
10 // Pode ou não ser abstrata
11 public class BaseClass
12 {
13     // Método Fixo
14     public void Func()
15     {
16         SubFuncA();
17         SubFuncB();
18         SubFuncC();
19     }
20
21     protected virtual void SubFuncA()
22     {
23         Console.Write("Olá, ");
24     }
25
26     protected virtual void SubFuncB()
27     {
28         Console.Write("Mundo");
29     }
30
31     protected virtual void SubFuncC()
32     {
33         Console.WriteLine("!");
34     }
```

```
35  }
36
37  public class ImplA : BaseClass
38  {
39      protected override void SubFuncA()
40      {
41          Console.Write("Hello, ");
42      }
43
44      protected override void SubFuncB()
45      {
46          Console.Write("World");
47      }
48  }
49
50  public class ImplB : BaseClass
51  {
52      protected override void SubFuncA()
53      {
54          Console.Write("Olá... ");
55      }
56
57      protected override void SubFuncC()
58      {
59          Console.WriteLine("?");
60      }
61  }
```

4 Strategy

O Strategy permite que criemos um contexto que pode mudar o seu comportamento conforme você define as estratégias dentro da classe.

```
1  using System;
2  using static System.Console;
3
4  Context context = new Context();
5
6  context.SetStrategy(new ConcreteStrategyA());
7  WriteLine(context.Compute(2, 3));
8
9  context.SetStrategy(new ConcreteStrategyB());
10 WriteLine(context.Compute(2, 3));
11
12 context.SetStrategy(new ConcreteStrategyC());
13 WriteLine(context.Compute(2, 3));
14
15 public interface IStrategy
16 {
17     double MakeOperation(double a, double b);
18 }
19
20 public class ConcreteStrategyA : IStrategy
21 {
22     public double MakeOperation(double a, double b)
23         => a + b;
24 }
25
26 public class ConcreteStrategyB : IStrategy
27 {
28     public double MakeOperation(double a, double b)
29         => a * b;
30 }
31
32 public class ConcreteStrategyC : IStrategy
33 {
34     public double MakeOperation(double a, double b)
35         => Math.Pow(a, b);
```

```
36  }
37
38  public class Context
39  {
40      private IStrategy strategy;
41
42      public void SetStrategy(IStrategy strategy)
43          => this.strategy = strategy;
44
45      public double Compute(double a, double b)
46          => this.strategy.MakeOperation(a, b);
47  }
```

5 State

State é uma evolução do Strategy. A diferença é que o estado, diferente da estratégia, é apenas um estado que controla o contexto e pode decidir os próximos estados.

```
1  using System;
2  using static System.Console;
3
4  Context context = new Context();
5  context.ChangeState(new ConcreteStateA());
6
7  WriteLine(context.Compute(2, 3));
8  WriteLine(context.Compute(2, 3));
9  WriteLine(context.Compute(2, 3));
10
11 public class Context
12 {
13     private State state;
14
15     public void ChangeState(State state)
16     {
17         this.state = state;
18         this.state.SetContext(this);
19     }
20
21     public double Compute(double a, double b)
22         => this.state.MakeOperation(a, b);
23 }
24
25 public abstract class State
26 {
27     protected Context context = null;
28     public void SetContext(Context context)
29         => this.context = context;
30
31     public abstract double MakeOperation(double a, double b);
32 }
33
34 public class ConcreteStateA : State
```

```
35  {
36      public override double MakeOperation(double a, double b)
37      {
38          // Muda o Estado para o B
39          this.context.ChangeState(new ConcreteStateB());
40          return a + b;
41      }
42
43  }
44
45  public class ConcreteStateB : State
46  {
47      public override double MakeOperation(double a, double b)
48      {
49          // Muda o Estado para o C
50          this.context.ChangeState(new ConcreteStateC());
51          return a * b;
52      }
53  }
54
55  public class ConcreteStateC : State
56  {
57      public override double MakeOperation(double a, double b)
58      {
59          // Muda o Estado para o A
60          this.context.ChangeState(new ConcreteStateA());
61          return Math.Pow(a, b);
62      }
63  }
```

6 Exemplo: Bot para jogos digitais

Vamos praticar o Proxy, Template Method, Strategy e State através de um exemplo. Vamos fazer um Bot que você pode aplicar em muitos contextos como jogos, por exemplo.

7 Exercícios