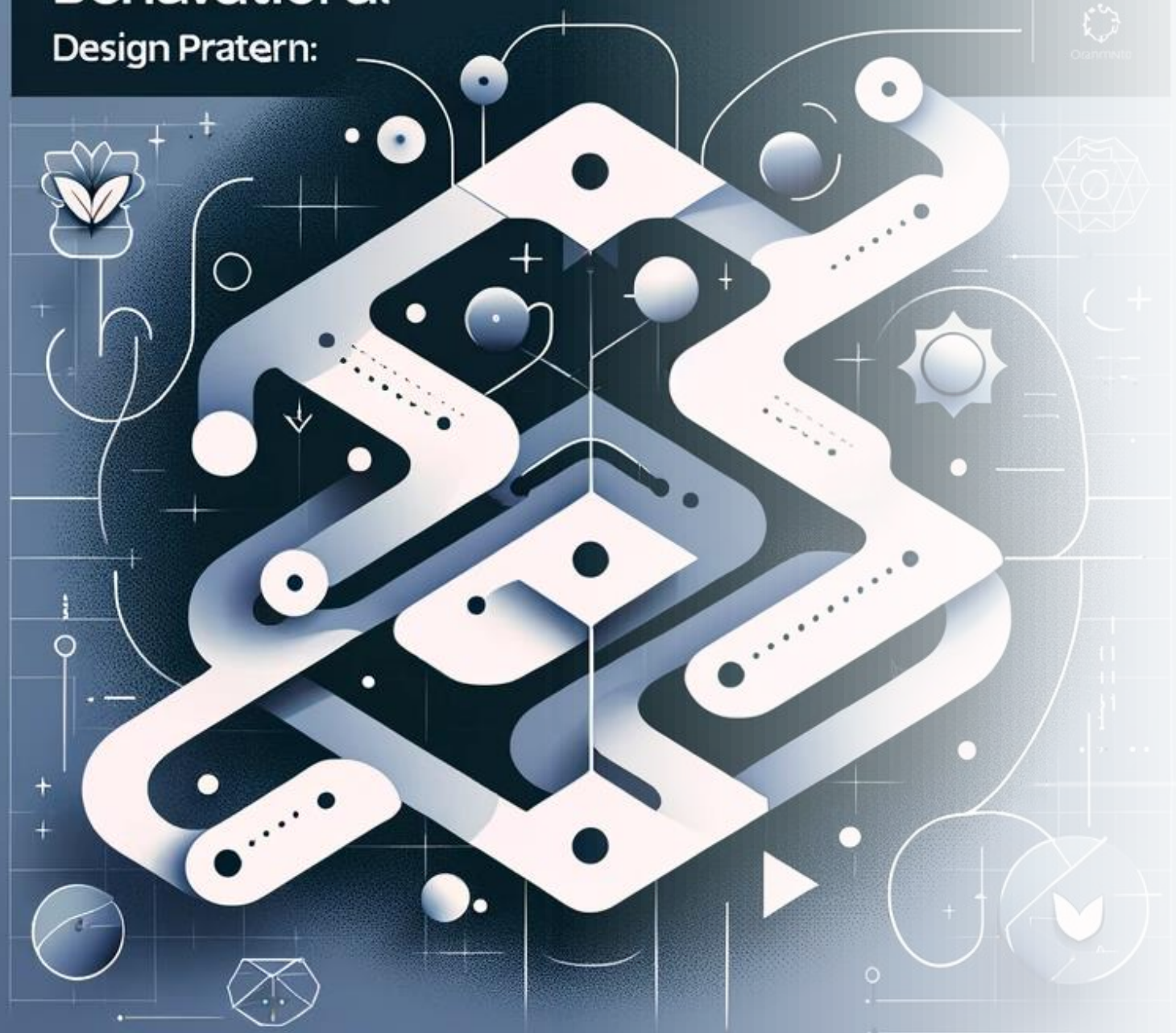


State

Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

O ***State*** permite que um objeto altere o seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado de classe.

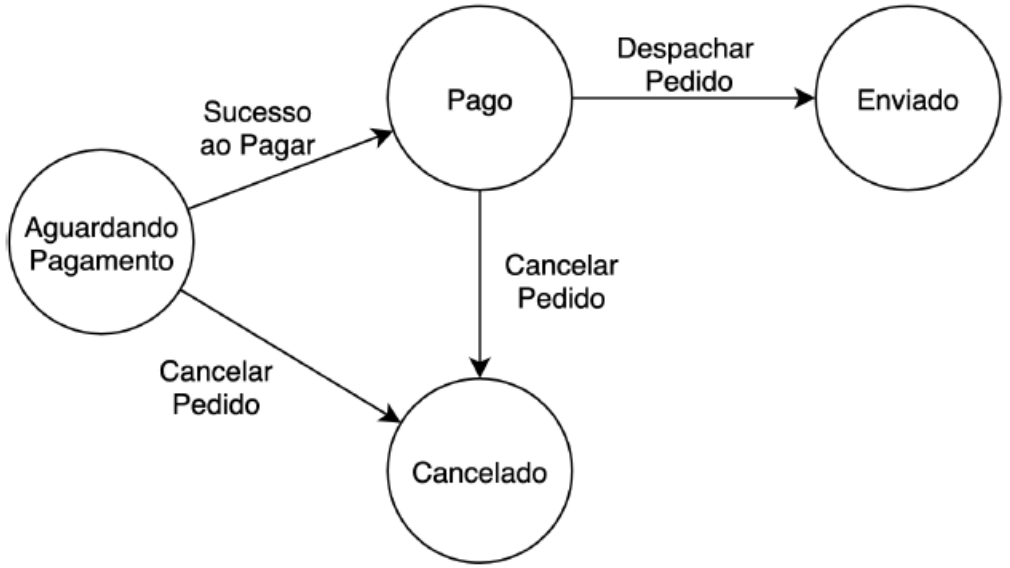
Em alguns contextos de desenvolvimento de software existem objetos que se comportam de forma diferente conforme o estado em que se encontram.

Nestes casos pode ser muito trabalhoso gerenciar tais mudanças de estado, além de gerar classes enormes, com diversas operações condicionais que definem se é possível migrar de um estado para o outro.

É comum encontrar classe com um atributo estado (status) que utilizam valores inteiros ou constantes para representar o estado interno em que um objeto se encontra.

Toda a lógica de transição entre estados costuma ficar na própria classe.

- Considere um pedido em um e-commerce onde tal pedido pode passar pelos estados:
  - Um pedido é inicializado no estado *Aguardando Pagamento.*
  - Um pedido pode ir para o estado *Pago*, se e somente se, estiver no estado *Aguardando Pagamento* e a ação *Sucesso ao Pagar* for solicitada.
  - Um pedido pode ir para o estado *Cancelado*, se e somente se, estiver nos estados *Aguardando Pagamento* ou *Pago* e a ação *Cancelar Pedido* for solicitada.
  - Um pedido pode ir para o estado *Enviado*, se e somente se estiver no estado *Pago* e ação *Despachar Pedido* por solicitada.



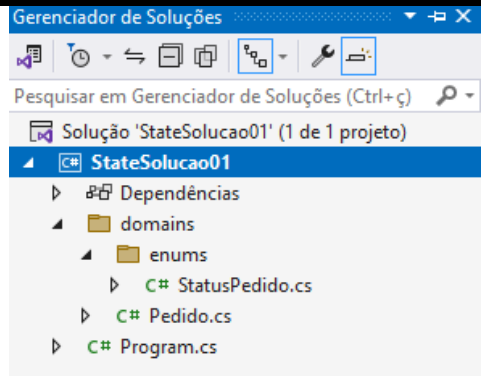
Máquina de estados

Classe Pedido

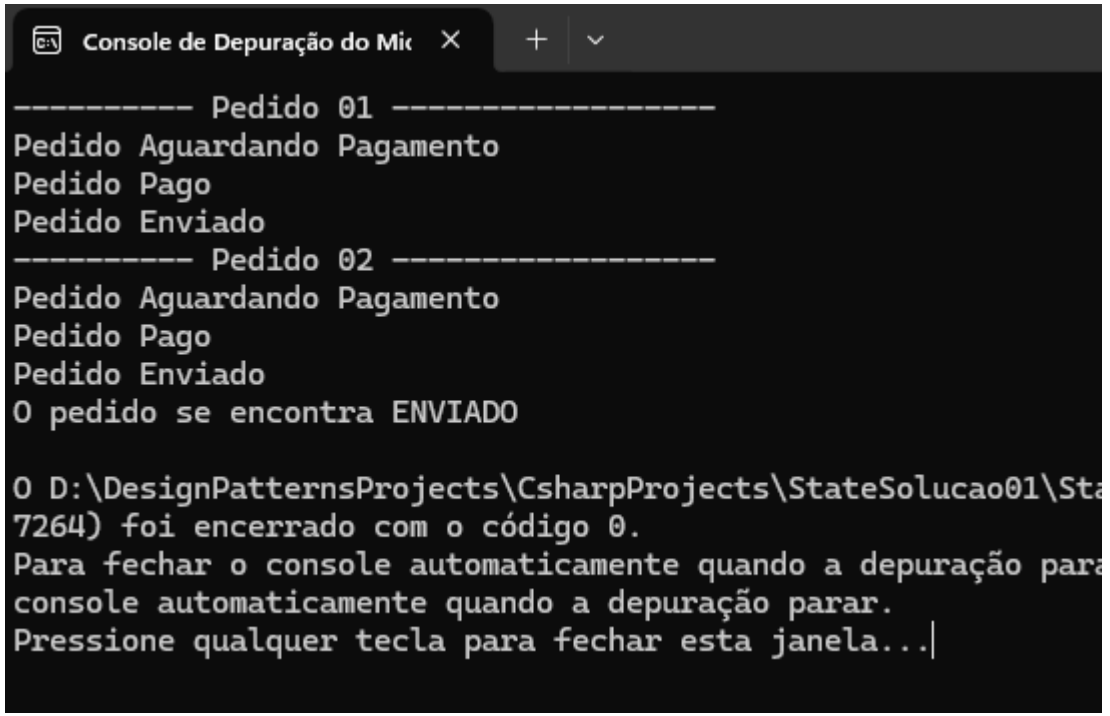
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

using StateSolucao01.domains.enums;

namespace StateSolucao01.domains
{
    5 referências
    public class Pedido
    {
        9 referências
        private StatusPedido statusPedido { get; set; }

        2 referências
        public Pedido() {
            Console.WriteLine("Pedido Aguardando Pagamento");
            this.statusPedido = StatusPedido.AGUARDANDO_PAGAMENTO;
        }

        2 referências
        public void sucessoAoPagar()
        {
            if (this.statusPedido == StatusPedido.AGUARDANDO_PAGAMENTO)
            {
                //muda status
                Console.WriteLine("Pedido Pago");
                this.statusPedido = StatusPedido.PAGO;
            }
            else
            {
                throw new Exception("O pedido não está aguardando pagamento");
            }
        }

        public void cancelarPedido()
        {
            if (this.statusPedido == StatusPedido.PAGO ||
                this.statusPedido == StatusPedido.AGUARDANDO_PAGAMENTO)
            {
                //muda status
                Console.WriteLine("Pedido Cancelado");
                this.statusPedido = StatusPedido.CANCELADO;
            }
            else
            {
                throw new Exception("O pedido se encontra "+this.statusPedido.ToString());
            }
        }

        2 referências
        public void despacharPedido()
        {
            if(this.statusPedido == StatusPedido.PAGO)
            {
                //muda status
                Console.WriteLine("Pedido Enviado");
                this.statusPedido = StatusPedido.ENVIADO;
            } else
            {
                throw new Exception("O pedido se encontra cancelado");
            }
        }

    }
}
```

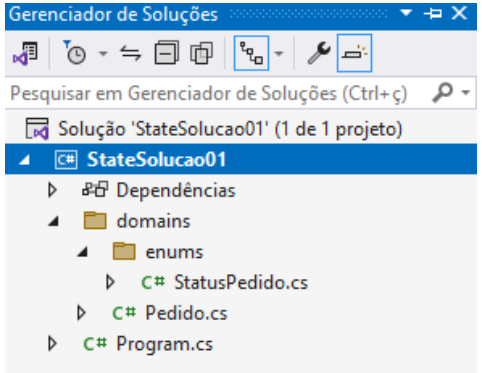
Enum→

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StateSolucao01.domains.enums
{
    9 referências
    enum StatusPedido
    {
        AGUARDANDO_PAGAMENTO=1,
        PAGO=2,
        CANCELADO=3,
        ENVIADO=4
    }
}
```



```
2   using StateSolucao01.domains;
3   try
4   {
5       Console.WriteLine("---------- Pedido 01 ----------------");
6       //Faça seus teste aqui!!!
7       Pedido pedido = new Pedido();
8       pedido.sucessoAoPagar();
9       pedido.despacharPedido();
10
11      Console.WriteLine("---------- Pedido 02 ----------------");
12      Pedido pedido2 = new Pedido();
13      pedido2.sucessoAoPagar();
14      pedido2.despacharPedido();
15      pedido2.cancelarPedido();
16
17
18  } catch (Exception e)
19  {
20      Console.WriteLine(e.Message);
21  }
22
```

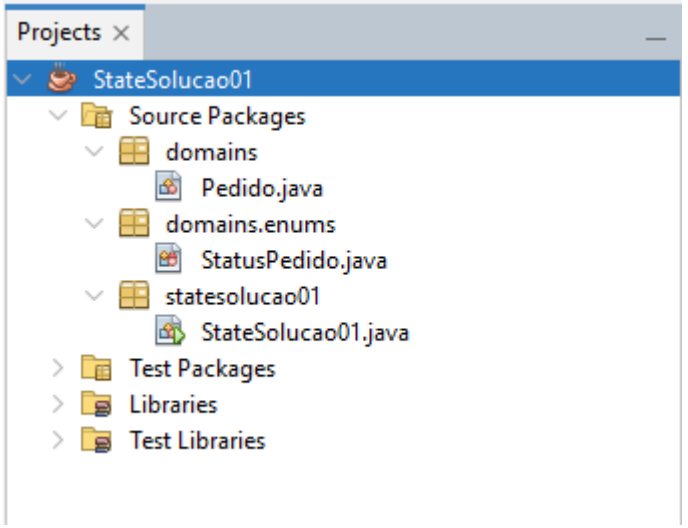




Faça seus Testes

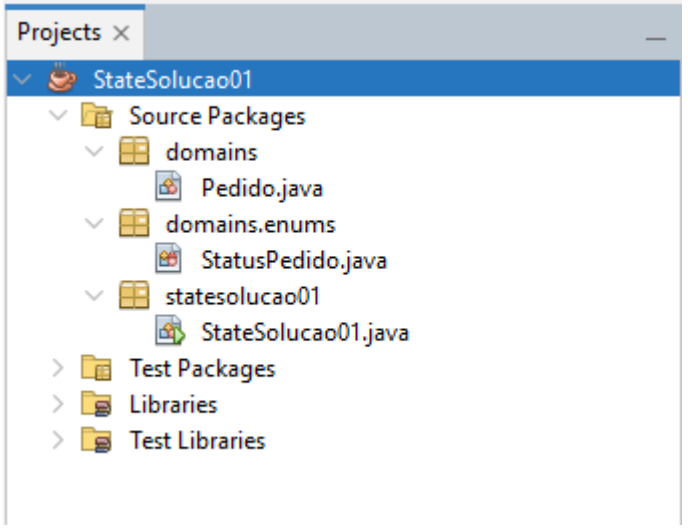
Enum

```java
...4 lines
package domains.enums;
/**...4 lines */
public enum StatusPedido {
    AGUARDANDO_PAGAMENTO(1,"AGUARDANDO_PAGAMENTO"),PAGO(2,"PAGO"),
    CANCELADO(2,"CANCELADO"),ENVIADO(3,"ENVIADO");

    private Integer id;
    private String statuPedido;

    private StatusPedido(Integer id, String statuPedido) {
        this.id = id;
        this.statuPedido = statuPedido;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getStatuPedido() {
        return statuPedido;
    }

    public void setStatuPedido(String statuPedido) {
        this.statuPedido = statuPedido;
    }

    @Override
    public String toString() {
        return "StatusPedido{" + "id=" + id + ", statuPedido=" + statuPedido + '}';
    }

    public static StatusPedido toEnum(Integer id) {
        if(id==null) return null;
        for(StatusPedido x : StatusPedido.values()){
            if(id.equals(x.getId())){
                return x;
            }
        }
        throw new IllegalArgumentException("Status Pedido inválido");
    }
}
```



Pedido

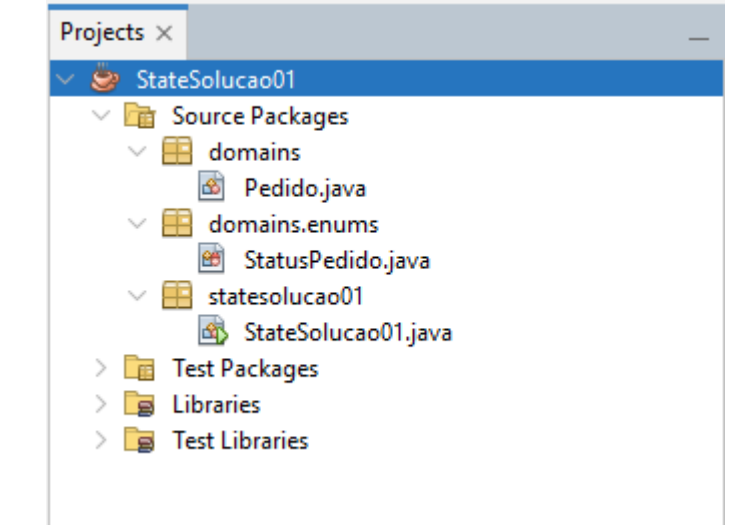
```java
...4 lines
package domains;

import domains.enums.StatusPedido;
/**
 *
 * @author jeffe
 */
public class Pedido {

    private StatusPedido statusPedido;

    public Pedido() {
        this.statusPedido = StatusPedido.AGUARDANDO_PAGAMENTO;
    }

    public void sucessoAoPagar() throws Exception
    {
        if (this.statusPedido == StatusPedido.AGUARDANDO_PAGAMENTO)
        {
            //muda status
            System.out.println("Pedido Pago");
            this.statusPedido = StatusPedido.PAGO;
        }
        else
        {
            throw new Exception("O pedido não está aguardando pagamento");
        }
    }

    public void cancelarPedido() throws Exception
    {
        if (this.statusPedido == StatusPedido.PAGO ||
            this.statusPedido == StatusPedido.AGUARDANDO_PAGAMENTO)
        {
            //muda status
            System.out.println("Pedido Cancelado");
            this.statusPedido = StatusPedido.CANCELADO;
        }
        else
        {
            throw new Exception("O pedido se encontra "+this.statusPedido.toString());
        }
    }
```

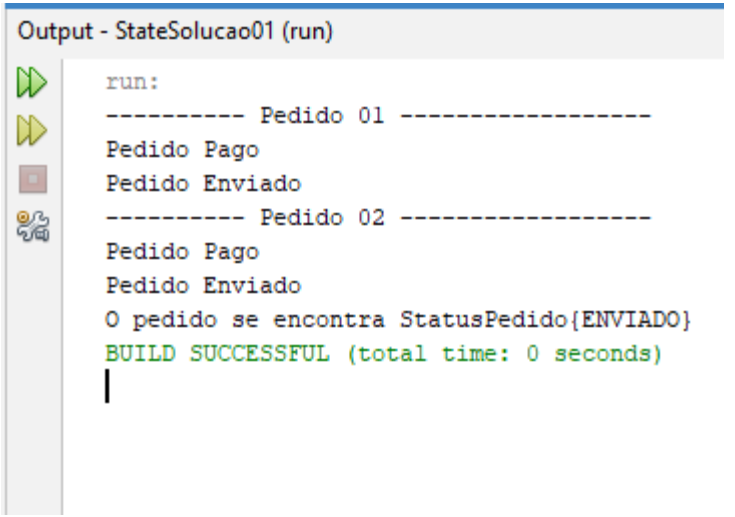



```java
    public void despacharPedido() throws Exception
    {
        if(this.statusPedido == StatusPedido.PAGO)
        {
            //muda status
            System.out.println("Pedido Enviado");
            this.statusPedido = StatusPedido.ENVIADO;
        } else
        {
            throw new Exception("O pedido se encontra cancelado");
        }
    }
}
```

```java
...4 lines
package statesolucao01;

import domains.Pedido;

/**...4 lines */
public class StateSolucao01 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try
        {
            System.out.println("---------- Pedido 01 ------------------");
            //Faça seus teste aqui!!!
            Pedido pedido = new Pedido();
            pedido.sucessoAoPagar();
            pedido.despacharPedido();

            System.out.println("---------- Pedido 02 ------------------");
            Pedido pedido2 = new Pedido();
            pedido2.sucessoAoPagar();
            pedido2.despacharPedido();
            pedido2.cancelarPedido();


        } catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }

}
```

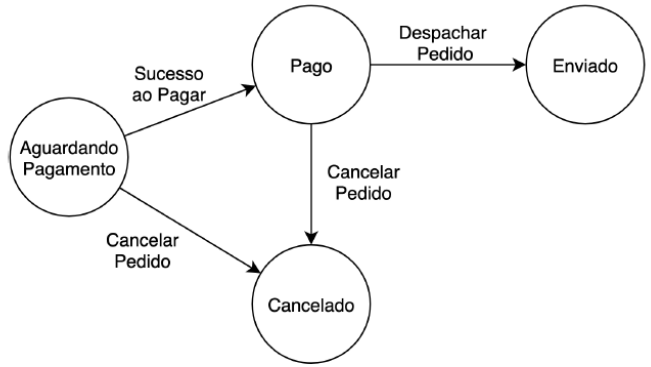
Faça seus Testes





- Problema dessa solução:
  - E se surgissem outros estados de transição no pedido como Aguardando Envio e outros;
  - Imagine quanto a classe que foi implementada não iria crescer e aumentar em complexidade para sua compreensão.
  - Geraria enums com mais itens.
  - Maior encadeamento de Ifs dentro da classe Pedido.
  - Assim o código apresentado não é muito escalável.
  - Entre outros problemas.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace StateSolucao01.domains.enums
8   {
        9 referências
9       enum StatusPedido
10      {
11          AGUARDANDO_PAGAMENTO=1,
12          PAGO=2,
13          CANCELADO=3,
14          ENVIADO=4
15      }
16  }
17
```



Máquina de estados

• O Padrão State – Quando Utilizar?

- Quando o comportamento de um objeto depende do seu estado interno, e com base nele muda seu comportamento em tempo de execução
- Quando operações possuírem instruções condicionais grandes que dependam do estado interno do objeto. Frequentemente várias destas operações terão as mesmas estruturas condicionais.

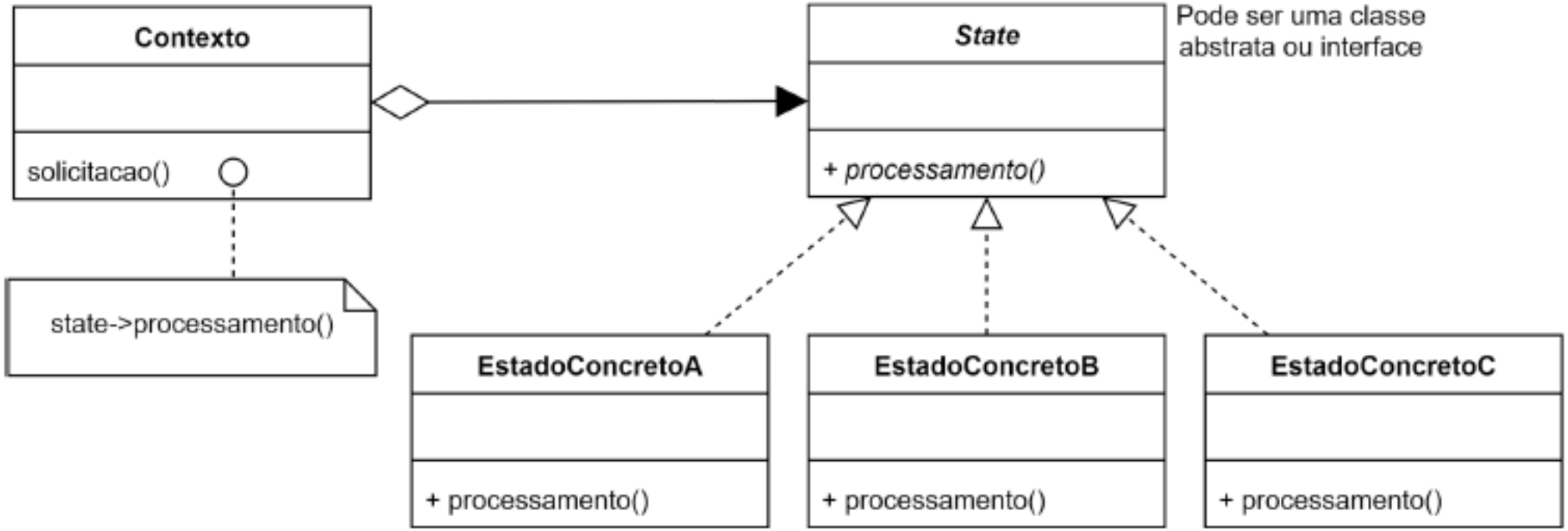


Diagrama de Classes

• O Padrão State – Componentes

- **Contexto**: é a classe que pode ter vários estados internos diferentes.
  - Ela mantém uma instância de uma subclasse **EstadoConcreto** que define seu estado interno atual.
  - Sempre que uma solicitação é feita ao **Contexto**, ela é delegada ao *estado atual* para ser processada.

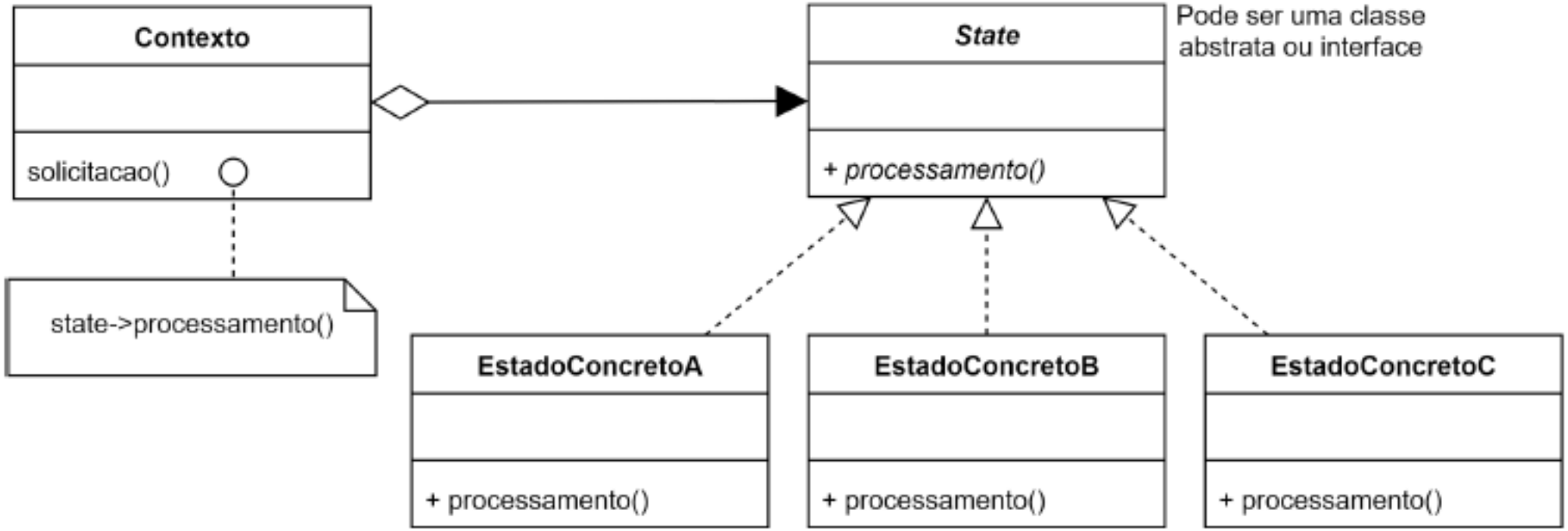


Diagrama de Classes

• O Padrão State – Componentes

- **State**: define uma interface (ou classe abstrata) comum para todos os estados concretos.
- **EstadoConcreto**: lidam com as solicitações provenientes do contexto.
  - Cada **EstadoConcreto** fornece a sua própria implementação de uma solicitação.
  - Deste modo, quando o **Contexto** muda de *estado interno* o seu comportamento também muda.

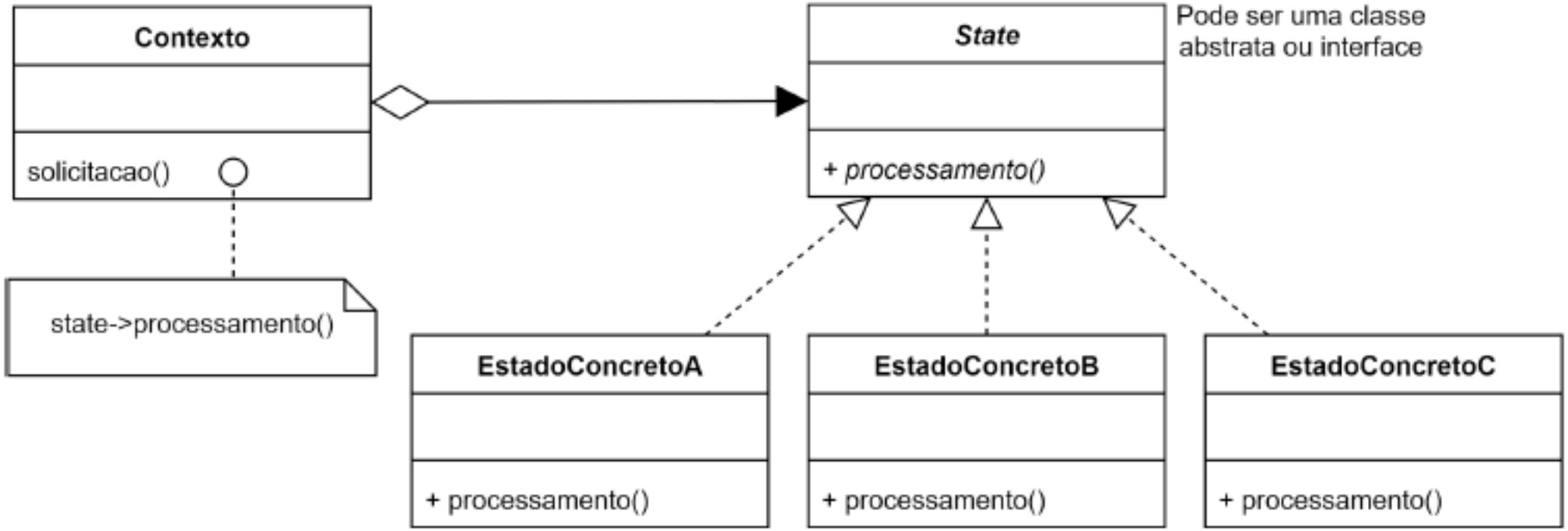
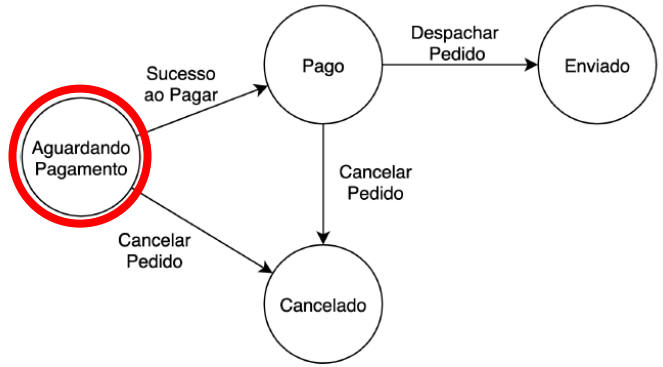


Diagrama de Classes

- Refatorando a aplicação
  - O padrão **State** sugere que cada um dos **estados se torne um objeto** que irá compor o objeto **Contexto** (instância da classe **Pedido**).
  - Um objeto **Contexto** é aquele que muda seu estado conforme a situação em que se encontra, e por consequência muda seu comportamento.
  - Deste modo o objeto Contexto parece ser uma instância de outra classe (A classe Pedido parecerá ter mudado de código), porém, tal mudança acontece devido a alteração do objeto que o compõem.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace StateSolucao01.domains.enums
8   {
        9 referências
9       enum StatusPedido
10      {
11          AGUARDANDO_PAGAMENTO=1,
12          PAGO=2,
13          CANCELADO=3,
14          ENVIADO=4
15      }
16  }
17
```



Máquina de estados

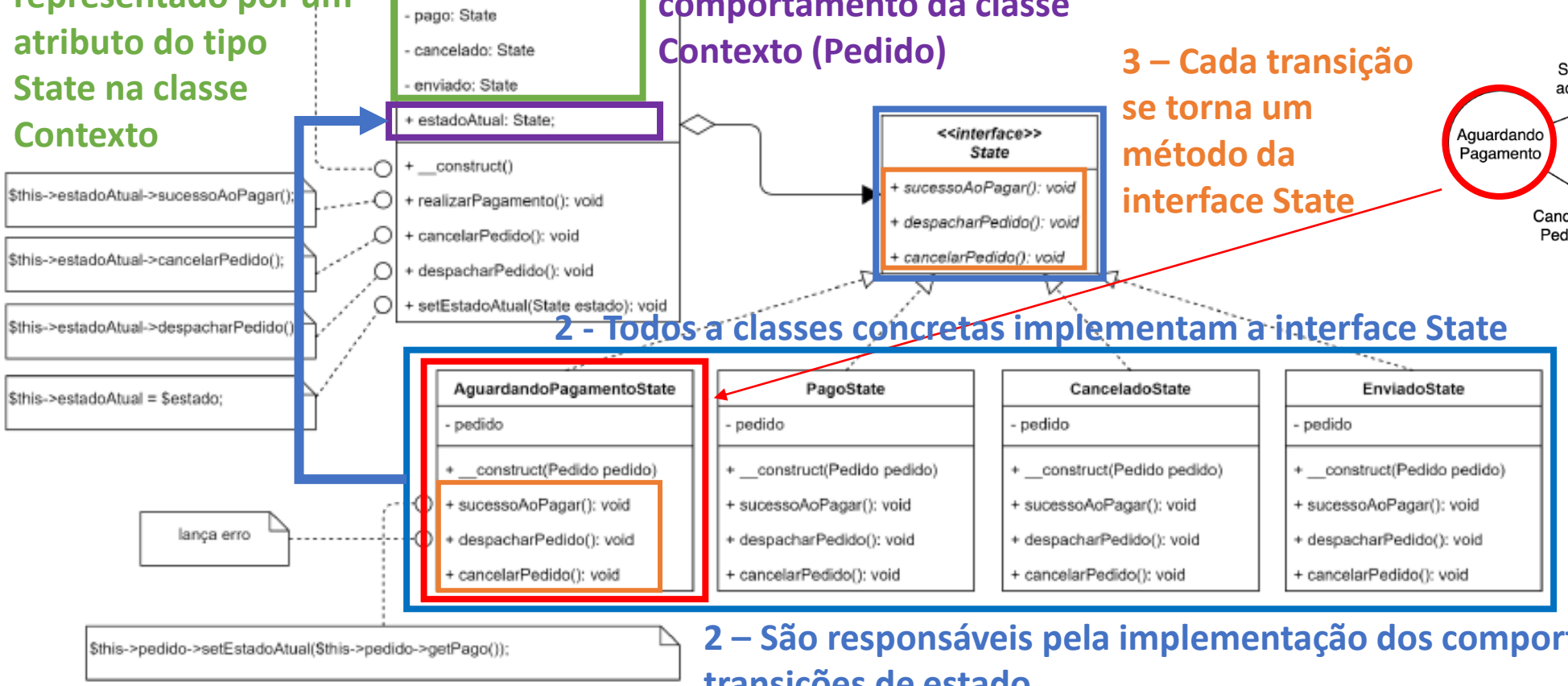
```
$this->aguardandoPagamento = new AguardandoPagamentoState($this);
$this->pago = new PagoState($this);
$this->cancelado = new CanceladoState($this);
$this->enviado = new EnviadoState($this);
$this->estadoAtual =  $this->aguardandoPagamento;
```

4 - Cada estado é representado por um atributo do tipo State na classe Contexto

5 – Atributo estadoAtual (tipo State) define o atual estado de comportamento da classe Contexto (Pedido)

3 – Cada transição se torna um método da interface State

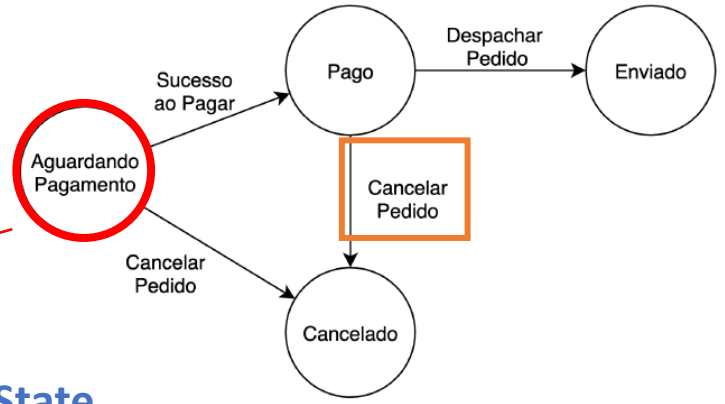
1 - Cada estado é representado em uma classe



2 - Todos a classes concretas implementam a interface State

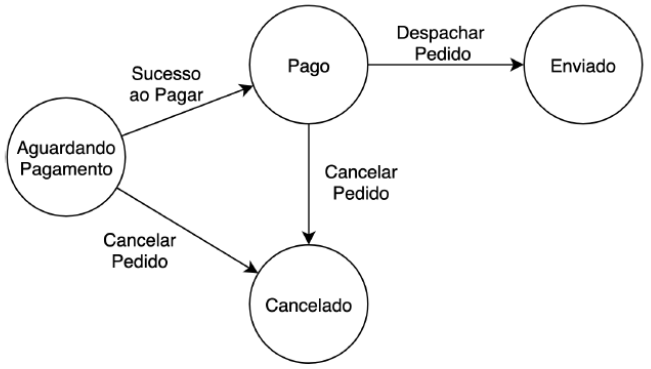
2 – São responsáveis pela implementação dos comportamentos e de suas possíveis transições de estado

Diagrama de classes do exemplo (Getters foram ocultados)



Máquina de estados

- Consequências
- O padrão **State** encapsula o comportamento específico de um estado, e como o objeto de contexto deve se comportar em cada estado.
  - O padrão coloca todo o comportamento associado a um estado específico em um objeto separado, assim, todo código referente a tal comportamento fica em uma subclasse **EstadoConcreto**.
  - Novos estados podem ser adicionados facilmente, apenas definindo novas subclasses **EstadoConcreto**.
- As transições de estado se tornam explícitas
  - Quando um objeto define seu estado atual apenas em termos de valores de dados internos (constantes ou inteiros), suas transições de estado não tem representação especifica.
  - Tais valores aparecem somente como atribuições para variáveis.
- Estados podem proteger seu contexto interno de transições de inconsistências, porque as transições são processadas à nível de estado e não a nível de contexto.



Máquina de estados

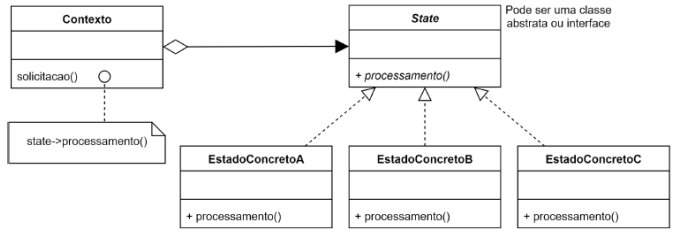
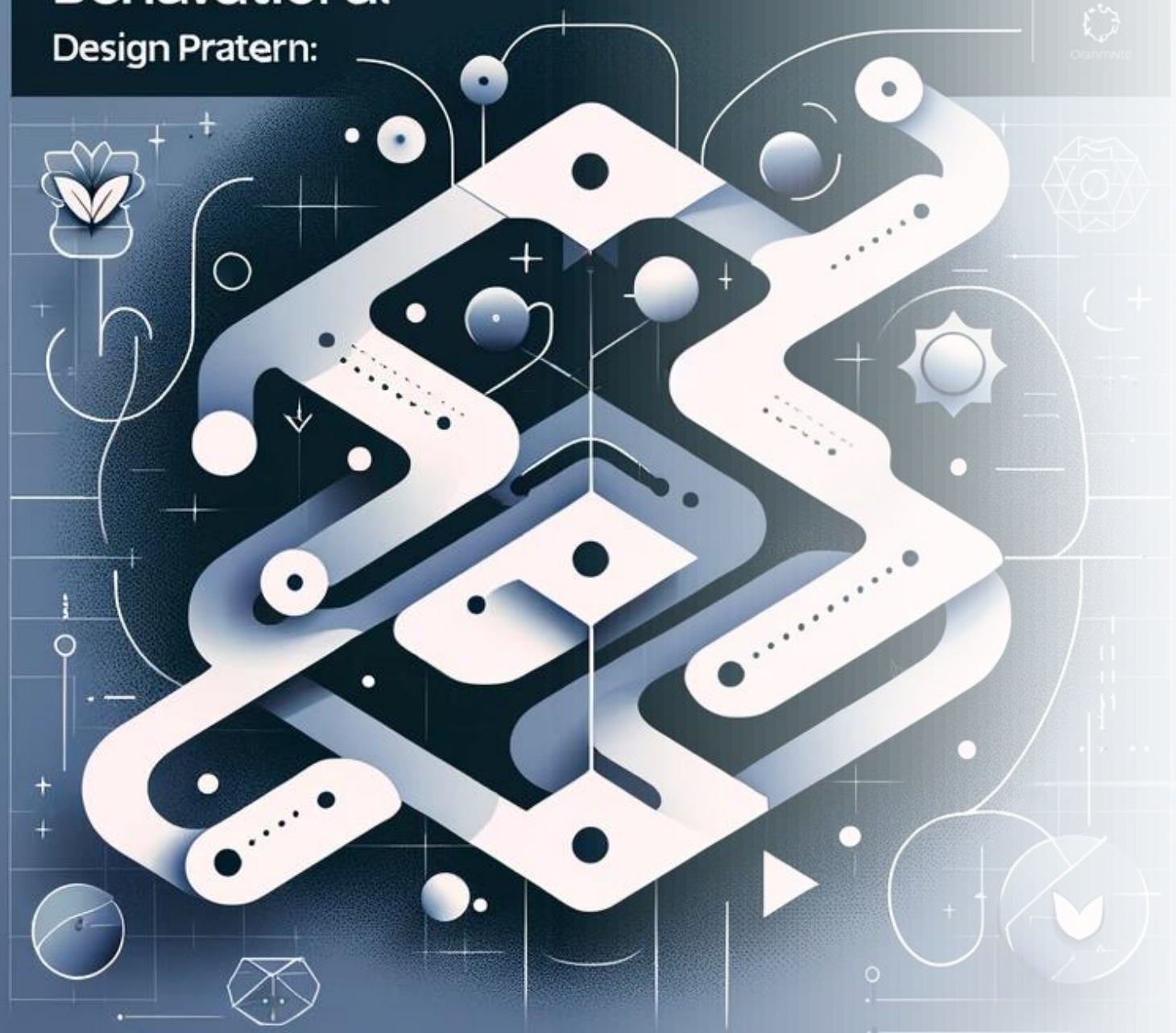


Diagrama de Classes

- **Relações com outros padrões**
- O **Bridge, State, Strategy** (e de certa forma o **Adapter**) têm estruturas muito parecidas. De fato, todos esses padrões estão baseados em composição, o que é delegar o trabalho para outros objetos.
  - Contudo, eles todos resolvem problemas diferentes. Um padrão não é apenas uma receita para estruturar seu código de uma maneira específica. Ele também pode comunicar a outros desenvolvedores o problema que o padrão resolve.

- **Relações com outros padrões**
- O **State** pode ser considerado como uma extensão do **Strategy**.
  - Ambos padrões são baseados em composição: eles mudam o comportamento do contexto ao delegar algum trabalho para objetos auxiliares.
  - O Strategy faz esses objetos serem completamente independentes e alheios entre si. Contudo, o State não restringe dependências entre estados concretos, permitindo que eles alterem o estado do contexto à vontade.

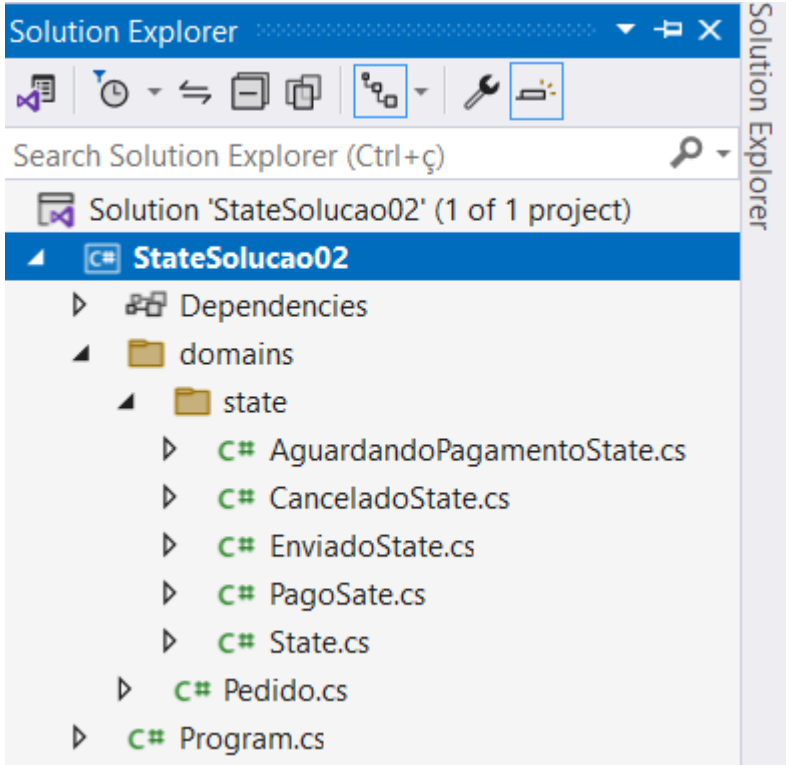


Implementação State – C# Solução 02

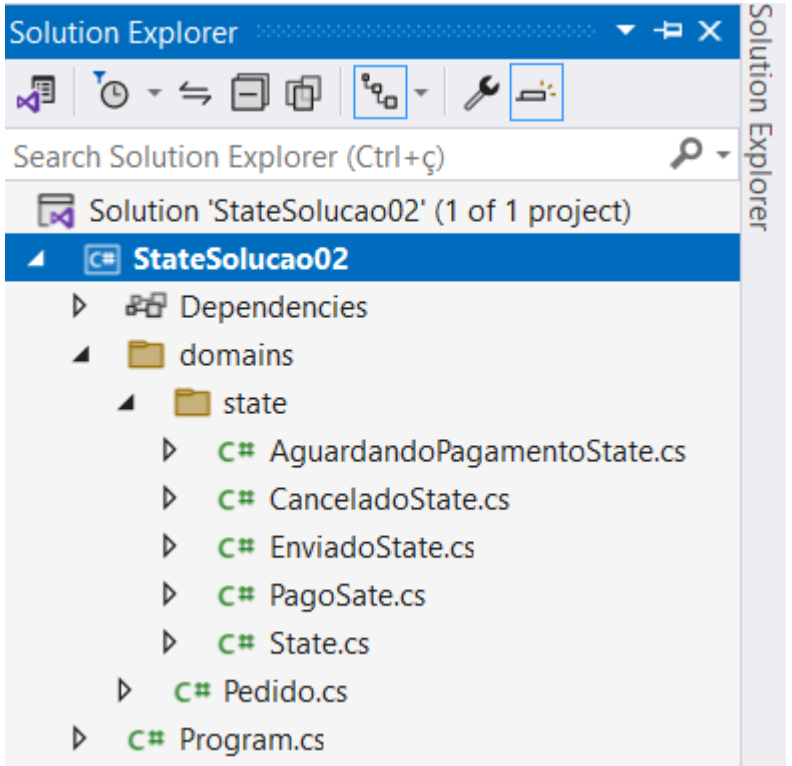
Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace StateSolucao02.domains.state
8  {
       21 references
9      public interface State
10     {
           5 references
11         public void sucessoAoPagar();
           5 references
12         public void cancelarPedido();
           5 references
13         public void despacharPedido();
14
15     }
16 }
17
```



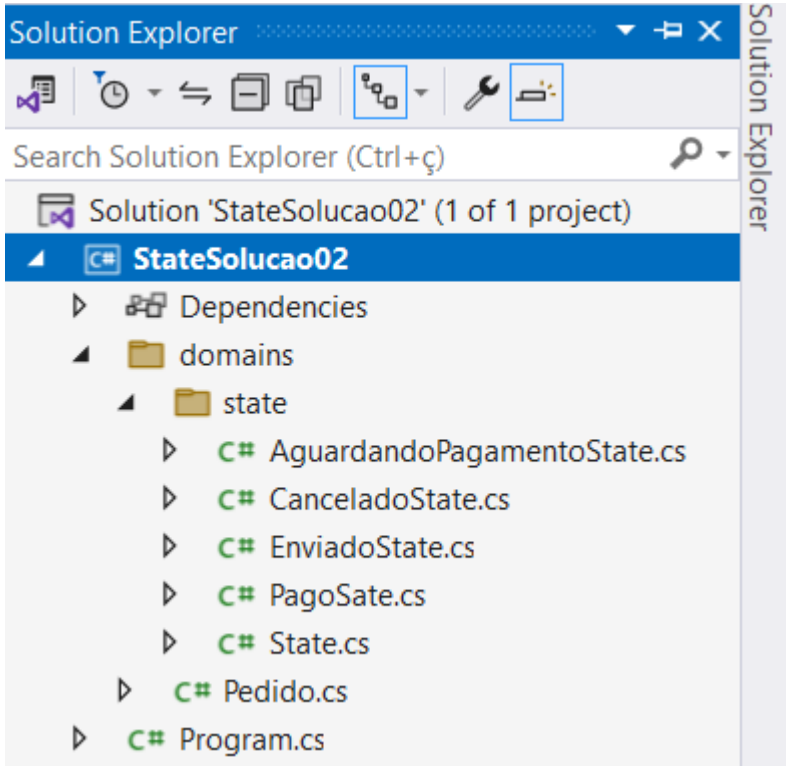
```
1  using StateSolucao02.domains.state;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using System.Xml.Serialization;
8
9  namespace StateSolucao01.domains
10 {
       13 references
11     public class Pedido
12     {
           2 references
13         public State aguardandoPagamento { get; set; }
           2 references
14         public State pago { get; set; }
           3 references
15         public State cancelado { get; set; }
           2 references
16         public State enviado { get; set; }
           8 references
17         public State estadoAtual { get; set; }
18
           2 references
19         public Pedido() {
20             Console.WriteLine("Pedido Aguardando Pagamento");
21             //define as classes concretas
22             this.aguardandoPagamento = new AguardandoPagamentoState(this);
23             this.pago = new PagoSate(this);
24             this.cancelado = new CanceladoState(this);
25             this.enviado = new EnviadoState(this);
26             //define estado atual
27             this.estadoAtual = this.aguardandoPagamento;
28         }
29
```



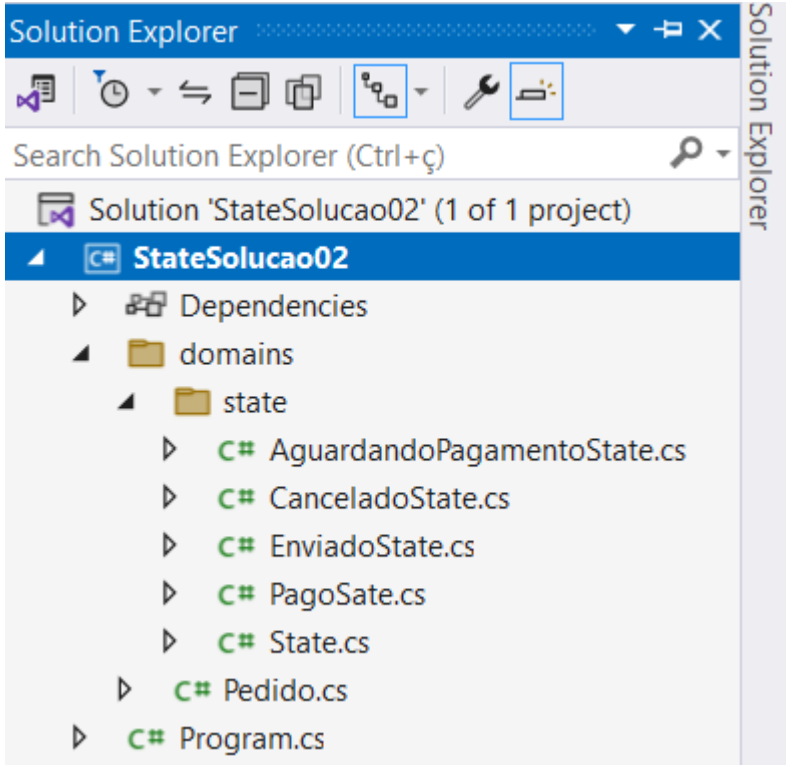
```
30          public void sucessoAoPagar()
31          {
32              try
33              {
34                  Console.WriteLine("Pedido Pago");
35                  this.estadoAtual.sucessoAoPagar();
36              } catch (Exception ex)
37              {
38                  Console.WriteLine(ex.Message);
39              }
40          }
41
            1 reference
42          public void cancelarPedido()
43          {
44              try
45              {
46                  Console.WriteLine("Pedido Cancelado");
47                  this.estadoAtual.cancelarPedido();
48              }
49              catch (Exception ex)
50              {
51                  Console.WriteLine(ex.Message);
52              }
53          }
54
```

```
55          public void despacharPedido()
56          {
57              try
58              {
59                  Console.WriteLine("Pedido enviado");
60                  this.estadoAtual.despacharPedido();
61              }
62              catch (Exception ex)
63              {
64                  Console.WriteLine(ex.Message);
65              }
66          }
67
68      }
69  }
70
```

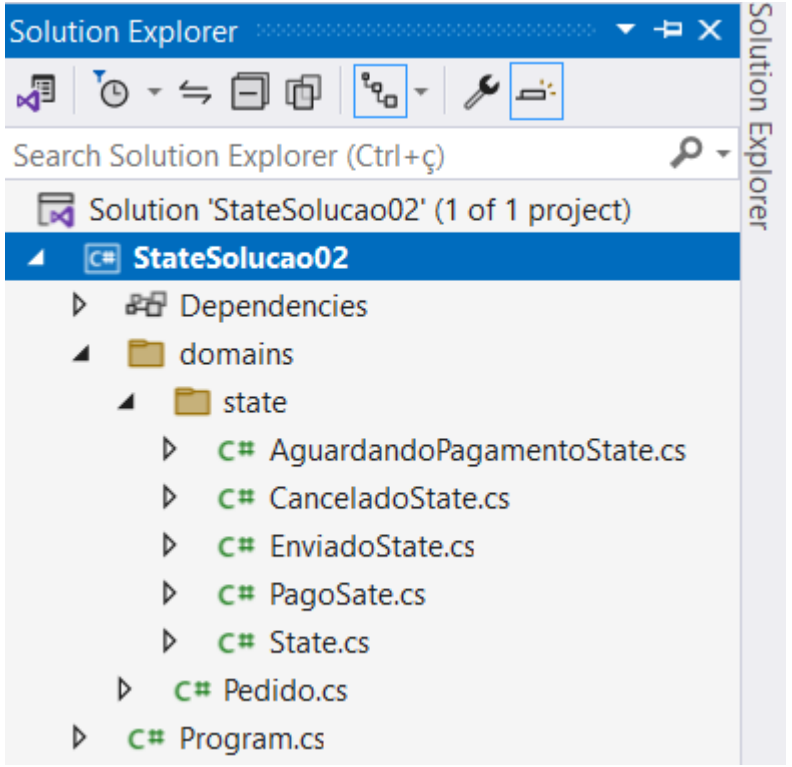
```
1   using StateSolucao01.domains;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace StateSolucao02.domains.state
9   {
        2 references
10      internal class AguardandoPagamentoState : State
11      {
12          private Pedido pedido;
            1 reference
13          public AguardandoPagamentoState(Pedido pedido)
14          {
15              this.pedido = pedido;
16          }
            2 references
17          void State.cancelarPedido()
18          {
19              this.pedido.estadoAtual = pedido.cancelado;
20          }
            2 references
21          void State.despacharPedido()
22          {
23              throw new Exception("Operacao não suportada, " +
24                  "pedido ainda não foi pago");
25          }
            2 references
26          void State.sucessoAoPagar()
27          {
28              this.pedido.estadoAtual = pedido.pago;
29          }
30      }
31  }
```



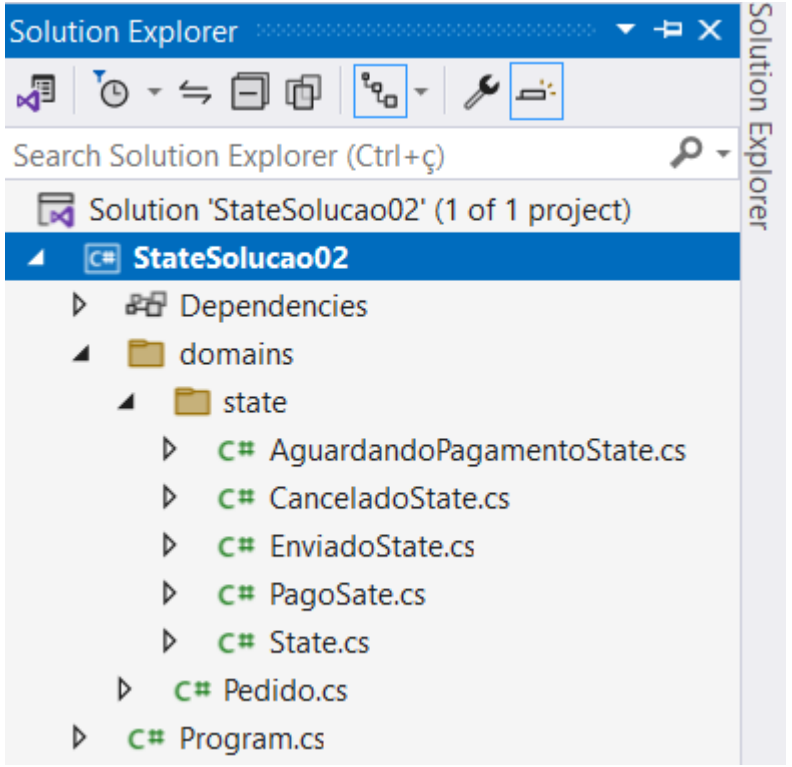
```
1   using StateSolucao01.domains;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace StateSolucao02.domains.state
9   {
        2 references
10      public class CanceladoState : State
11      {
12          private Pedido pedido;
            1 reference
13          public CanceladoState(Pedido pedido)
14          {
15              this.pedido = pedido;
16          }
            2 references
17          void State.cancelarPedido()
18          {
19              throw new Exception("Operacao não suportada, pedido foi cancelado");
20          }
            2 references
21          void State.despacharPedido()
22          {
23              throw new Exception("Operacao não suportada, pedido foi cancelado");
24          }
            2 references
25          void State.sucessoAoPagar()
26          {
27              throw new Exception("Operacao não suportada, pedido foi cancelado");
28          }
29      }
30  }
```



```
1   using StateSolucao01.domains;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace StateSolucao02.domains.state
9   {
        2 references
10      public class EnviadoState : State
11      {
12          private Pedido pedido;
            1 reference
13          public EnviadoState(Pedido pedido)
14          {
15              this.pedido = pedido;
16          }
            2 references
17          void State.cancelarPedido()
18          {
19              throw new Exception("Operacao não suportada, pedido já foi enviado");
20          }
            2 references
21          void State.despacharPedido()
22          {
23              throw new Exception("Operacao não suportada, pedido já foi enviado");
24          }
            2 references
25          void State.sucessoAoPagar()
26          {
27              throw new Exception("Operacao não suportada, pedido já foi enviado");
28          }
29      }
30  }
```

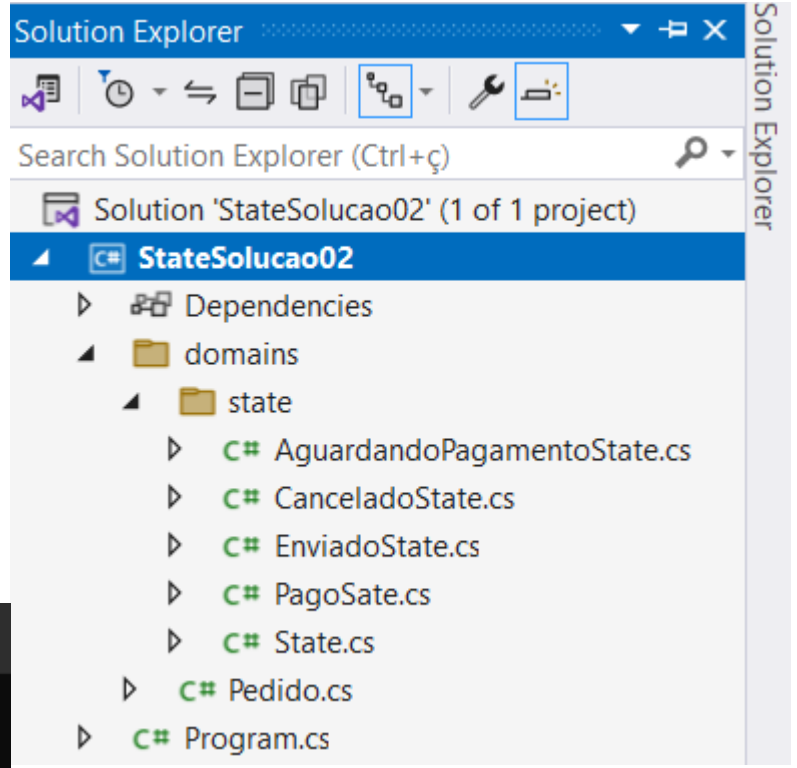


```
1  using StateSolucao01.domains;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace StateSolucao02.domains.state
9  {
       2 references
10     public class PagoSate : State
11     {
12         private Pedido pedido;
           1 reference
13         public PagoSate(Pedido pedido)
14         {
15             this.pedido = pedido;
16         }
           2 references
17         void State.cancelarPedido()
18         {
19             This method has 2 reference(s). (Alt+2)  = pedido.cancelado;
20         }
           2 references
21         void State.despacharPedido()
22         {
23             this.pedido.estadoAtual = pedido.enviado;
24         }
           2 references
25         void State.sucessoAoPagar()
26         {
27             throw new Exception("Operacao não suportada, pedido já foi pago");
28         }
29     }
30 }
```



```
using StateSolucao01.domains;

try
{
    Console.WriteLine("---------- Pedido 01 ----------------");
    //Faça seus teste aqui!!!
    Pedido pedido = new Pedido();
    pedido.sucessoAoPagar();
    pedido.despacharPedido();

    Console.WriteLine("---------- Pedido 02 ----------------");
    Pedido pedido2 = new Pedido();
    pedido2.sucessoAoPagar();
    pedido2.despacharPedido();
    pedido2.cancelarPedido();


}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

```
Microsoft Visual Studio Debu
---------- Pedido 01 ----------------
Pedido Aguardando Pagamento
Pedido Pago
Pedido enviado
---------- Pedido 02 ----------------
Pedido Aguardando Pagamento
Pedido Pago
Pedido enviado
Pedido Cancelado
Operacao não suportada, pedido já foi enviado

D:\Projetos\DesignPatternsProjects\CsharpProjec
cess 19116) exited with code 0.
To automatically close the console when debuggi
le when debugging stops.
Press any key to close this window . . .
```

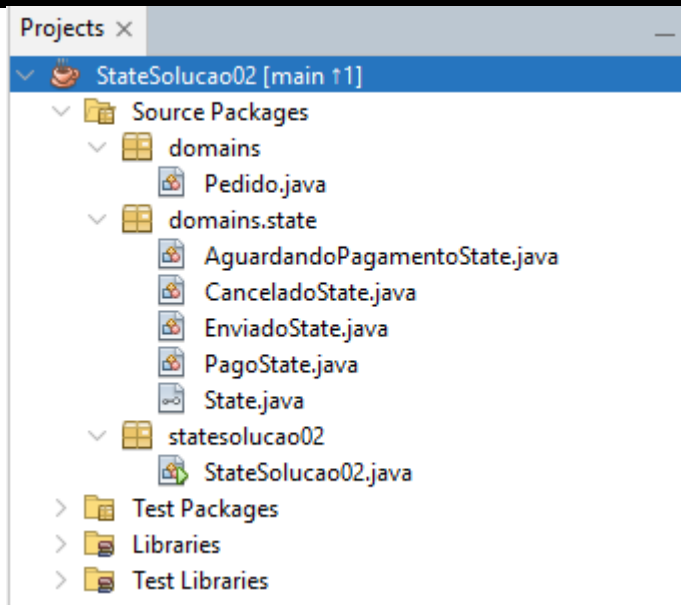


Implementação State – Java Solução 02

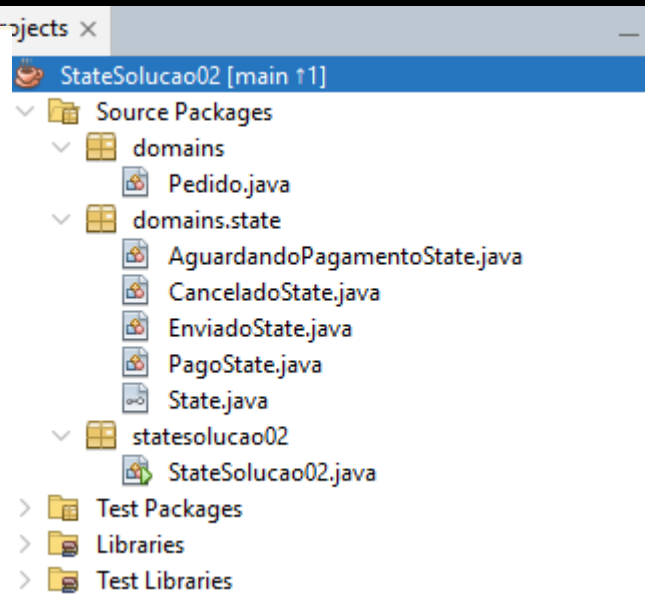
Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

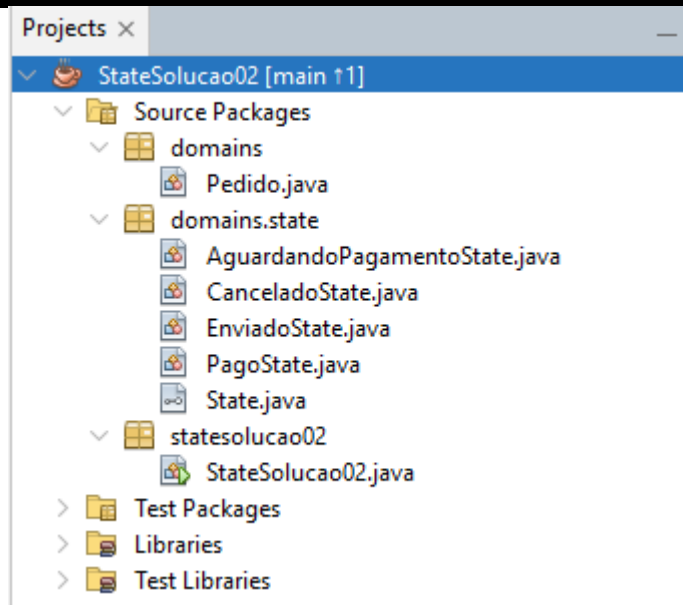
```
1   ...4 lines
5   package domains.state;
6
7   /**...4 lines */
    public interface State {
12
        public void sucessoAoPagar();
        public void cancelarPedido();
        public void despacharPedido();
16
17  }
```



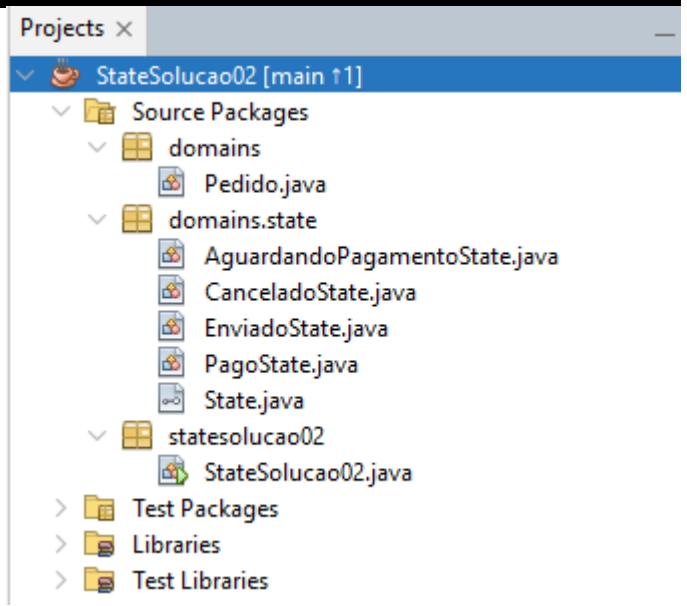
```java
...4 lines
package domains;

import domains.state.AguardandoPagamentoState;
import domains.state.CanceladoState;
import domains.state.EnviadoState;
import domains.state.PagoState;
import domains.state.State;

/**...4 lines */
public class Pedido {

    private State aguardandoPagamento;
    private State pago;
    private State cancelado;
    private State enviado;
    //estado atual do pedido.
    private State estadoAtual;

    public Pedido() {
        System.out.println("Pedido aguardando pagamento");
        this.aguardandoPagamento = new AguardandoPagamentoState(this);
        this.pago = new PagoState(this);
        this.cancelado = new CanceladoState(this);
        this.enviado = new EnviadoState(this);
        //define o estado atual
        this.estadoAtual = this.aguardandoPagamento;
    }

    public void sucessoAoPagar(){
        try{
            System.out.println("Pedido Pago");
            this.estadoAtual.sucessoAoPagar();
        } catch(Exception e){
            System.out.println(e.getMessage());
        }
    }

    public void cancelarPedido(){
        try{
            System.out.println("Pedido Cancelar");
            this.estadoAtual.cancelarPedido();
        } catch(Exception e){
            System.out.println(e.getMessage());
        }
    }

    public void despacharPedido(){
        try{
            System.out.println("Pedido Enviado");
            this.estadoAtual.despacharPedido();
        } catch(Exception e){
            System.out.println(e.getMessage());
        }
    }

    public State getAguardandoPagamento() {
        return aguardandoPagamento;
    }

    public State getCancelado() {
        return cancelado;
    }

    public State getEnviado() {
        return enviado;
    }

    public State getPago() {
        return pago;
    }

    public void setEstadoAtual(State estadoAtual) {
        this.estadoAtual = estadoAtual;
    }
}
```



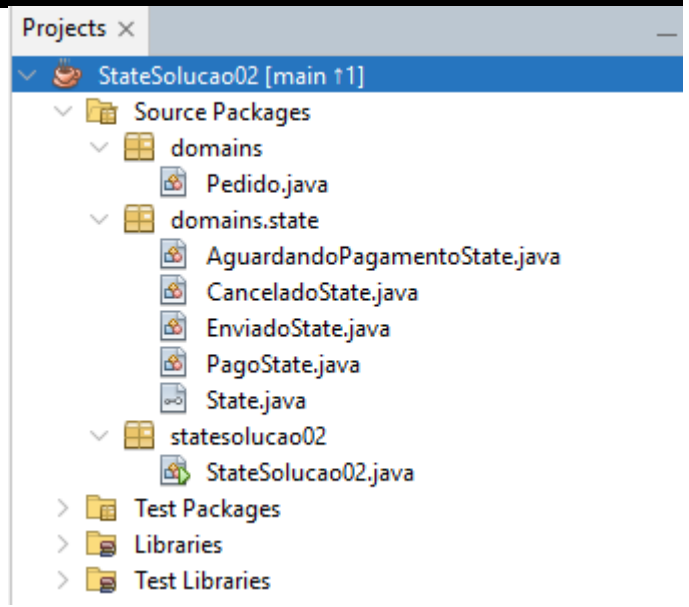
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class AguardandoPagamentoState implements State{

    private Pedido pedido;

    public AguardandoPagamentoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        this.pedido.setEstadoAtual(pedido.getPago());
    }

    @Override
    public void cancelarPedido() {
        this.pedido.setEstadoAtual(pedido.getCancelado());
    }

    @Override
    public void despacharPedido() {
        try {
            throw new Exception("Operação não suportada - pedido ainda não foi pago");
        } catch (Exception ex) {
            Logger.getLogger(AguardandoPagamentoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



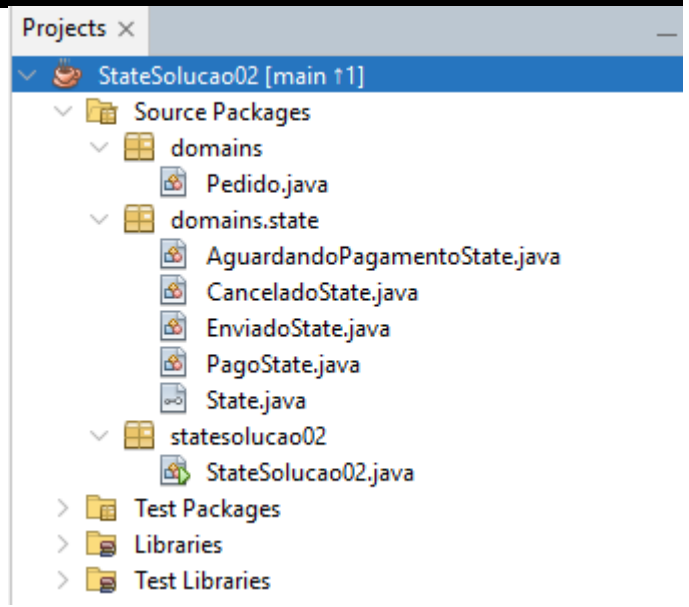
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class CanceladoState implements State{

    private Pedido pedido;

    public CanceladoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void cancelarPedido() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void despacharPedido() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



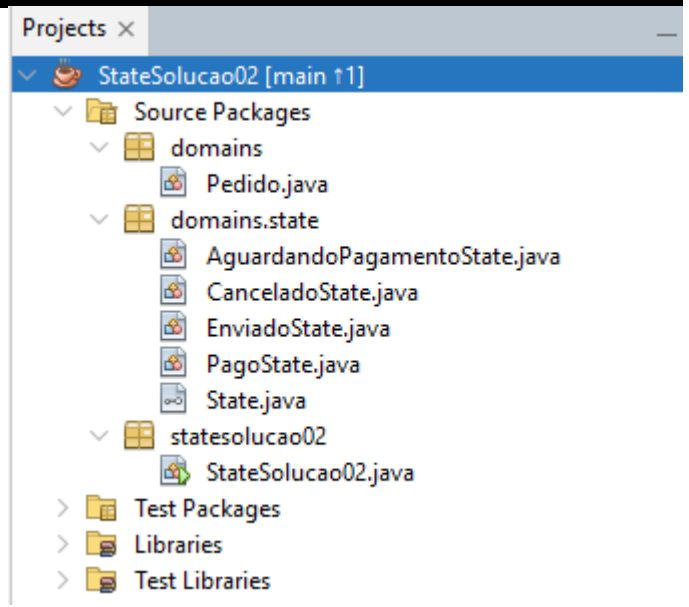
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class EnviadoState implements State{

    private Pedido pedido;

    public EnviadoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void cancelarPedido() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void despacharPedido() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



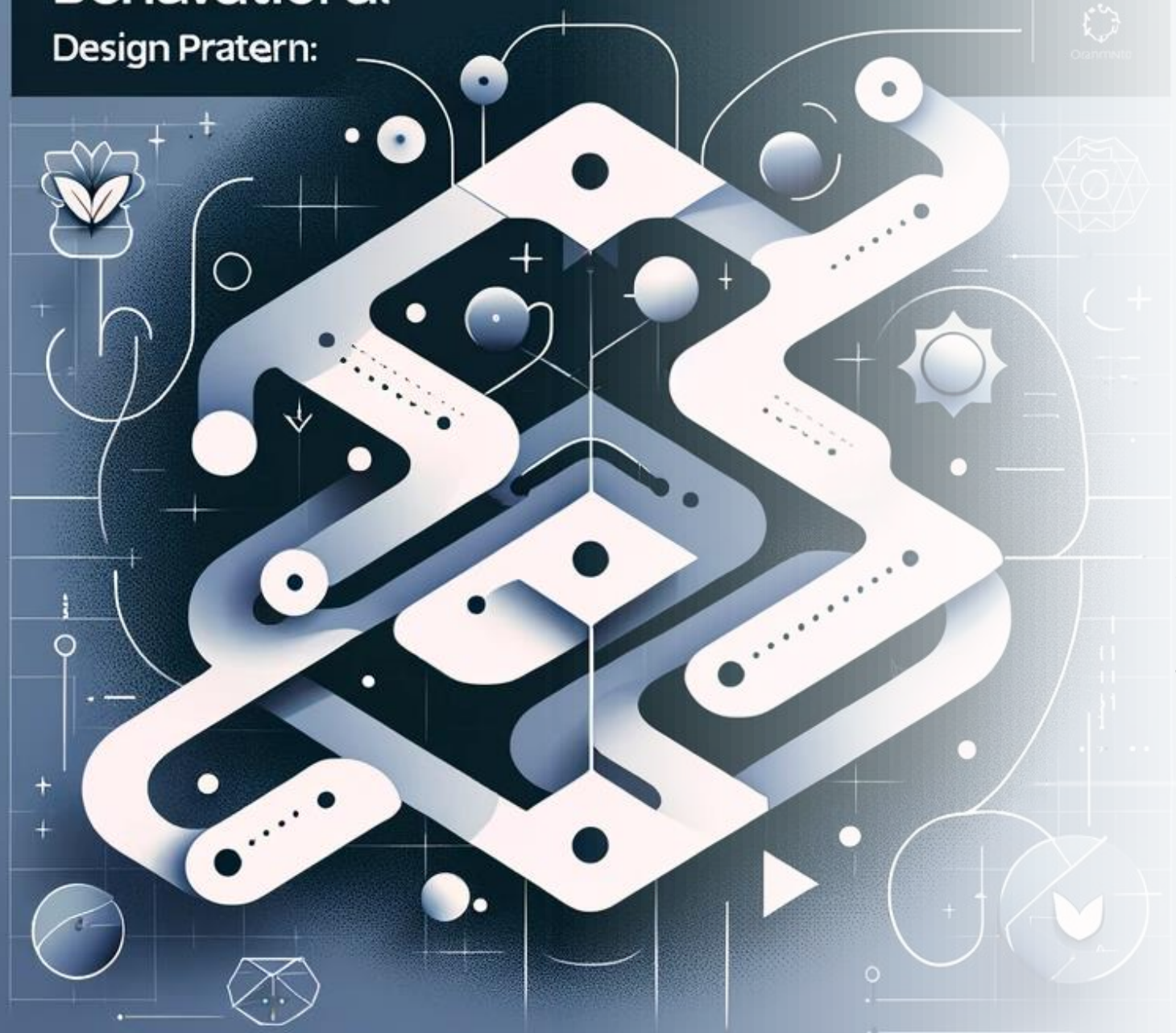
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class PagoState implements State{

    private Pedido pedido;

    public PagoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        try {
            throw new Exception("Operação não suportada - pedido já foi pago");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void cancelarPedido() {
        this.pedido.setEstadoAtual(pedido.getEnviado());
    }

    @Override
    public void despacharPedido() {
        this.pedido.setEstadoAtual(pedido.getEnviado());
    }

}
```



```java
...4 lines
package statesolucao02;

import domains.Pedido;

/**...4 lines */
public class StateSolucao02 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try
        {
            System.out.println("---------- Pedido 01 ------------------");
            //Faça seus teste aqui!!!
            Pedido pedido = new Pedido();
            pedido.sucessoAoPagar();
            pedido.despacharPedido();

            System.out.println("---------- Pedido 02 ------------------");
            Pedido pedido2 = new Pedido();
            pedido2.sucessoAoPagar();
            pedido2.despacharPedido();
            pedido2.cancelarPedido();


        } catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }

}
```




```
run:
---------- Pedido 01 ------------------
Pedido aguardando pagamento
Pedido Pago
Pedido Enviado
---------- Pedido 02 ------------------
Pedido aguardando pagamento
Pedido Pago
Pedido Enviado
Pedido Cancelar
mar. 21, 2024 4:27:32 PM domains.state.EnviadoState cancelarPedido
GRAVE: null
java.lang.Exception: Operação não suportada - pedido cancelado
        at domains.state.EnviadoState.cancelarPedido(EnviadoState.java:35)
        at domains.Pedido.cancelarPedido(Pedido.java:48)
        at statesolucao02.StateSolucao02.main(StateSolucao02.java:31)

BUILD SUCCESSFUL (total time: 0 seconds)
```



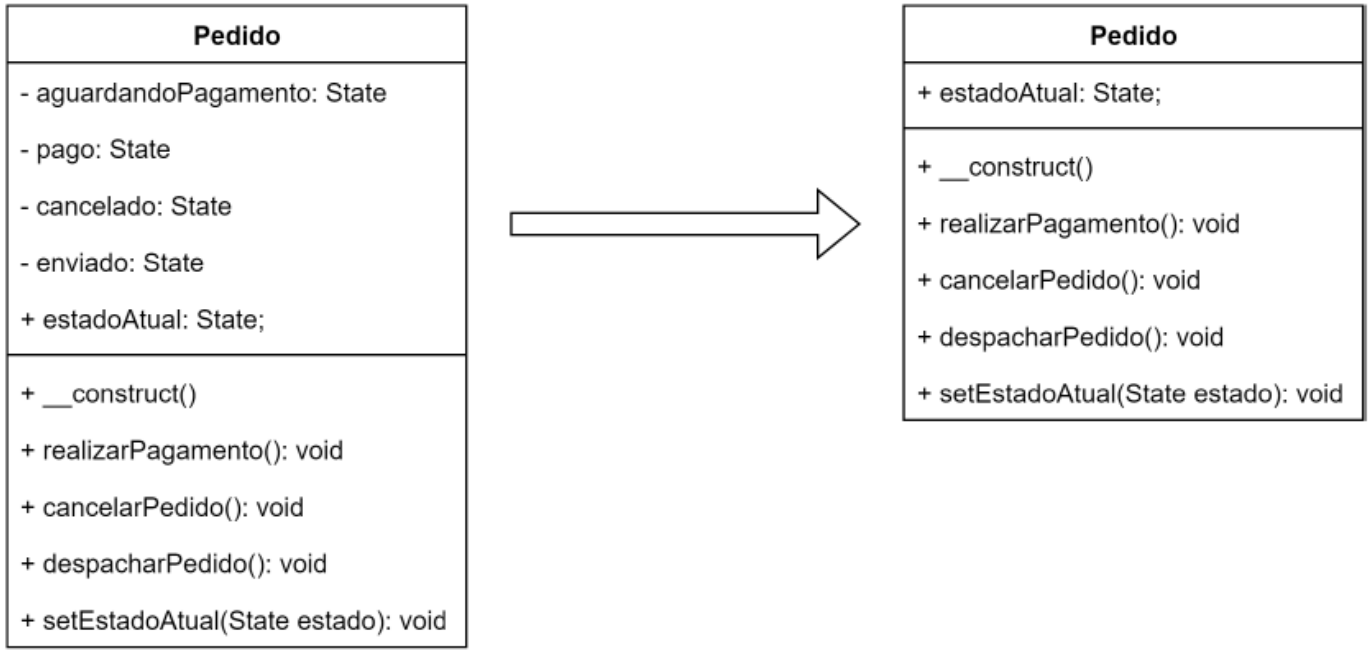
Implementação State – Solução 02 - Problemas

Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

O projeto da Solução 02 não respeita o princípio do Open/Closed do SOLID!

Se tivermos que inserir um novo estado como Finalizado teremos que atribuir novos atributos a classe Pedido necessitando de alterá-la.



Vamos melhorar nosso projeto para evitar de que a classe Pedido seja modificada.

Padrões de Projetos Comportamentais – State

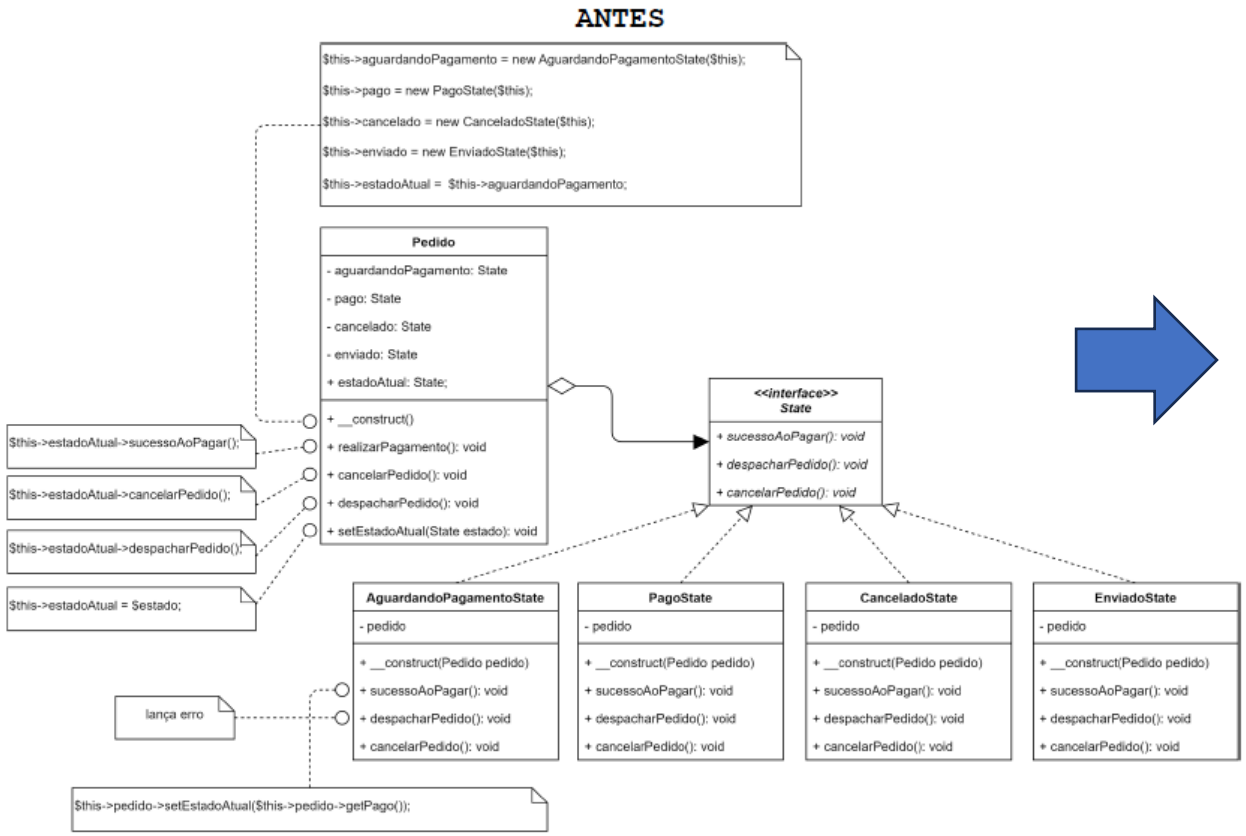


Diagrama de classes do exemplo (Getters foram ocultados)

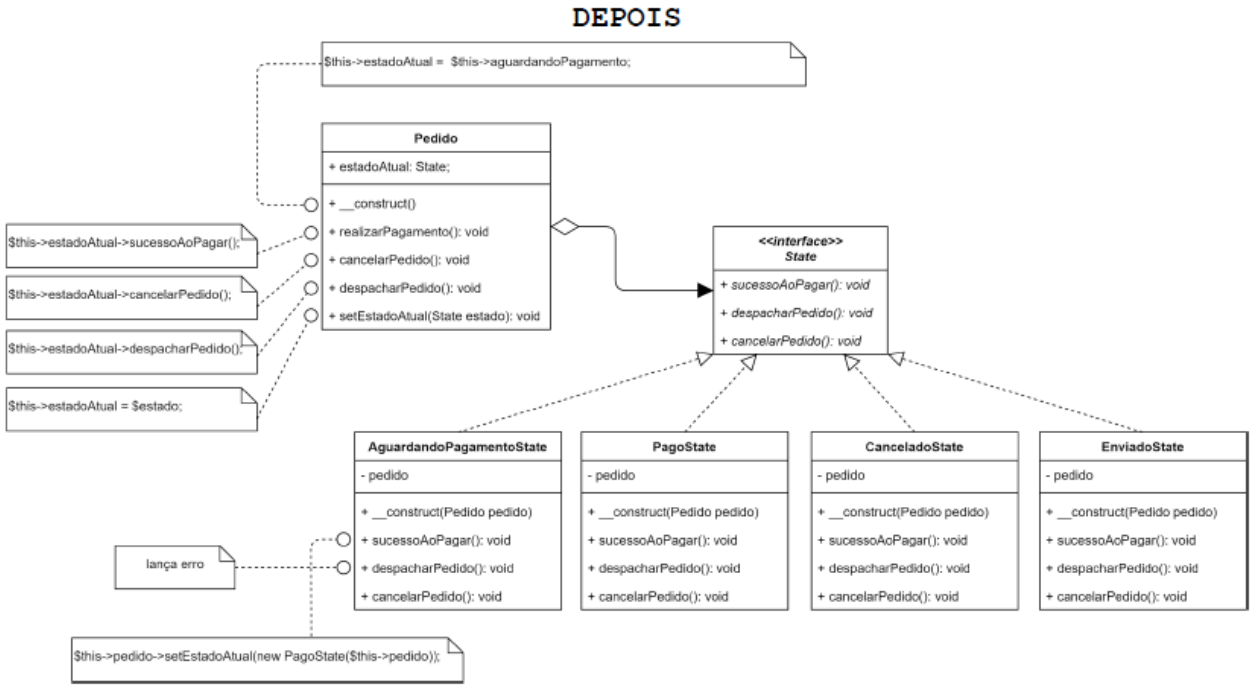
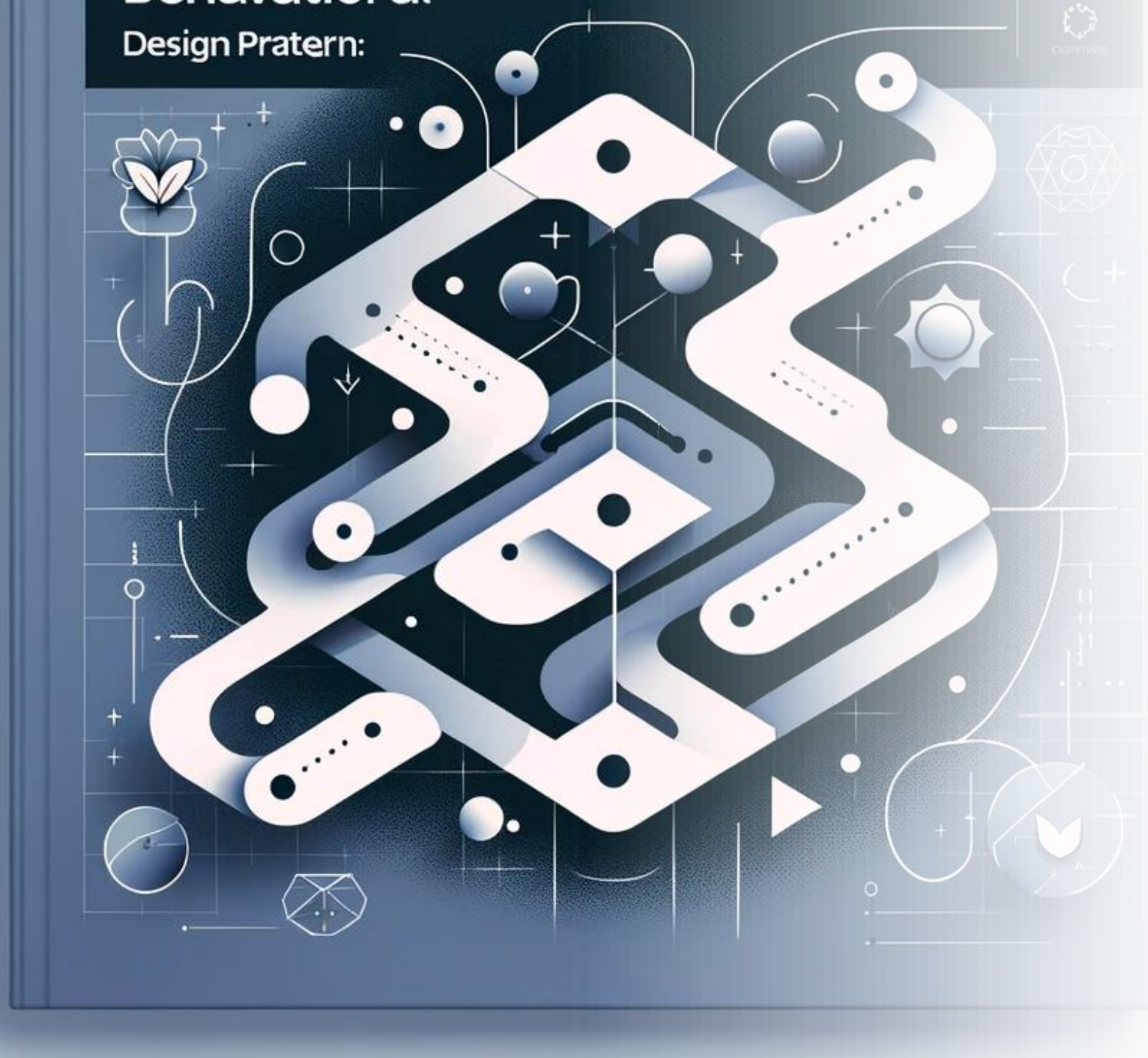


Diagrama de classes do exemplo (Getters foram ocultados)

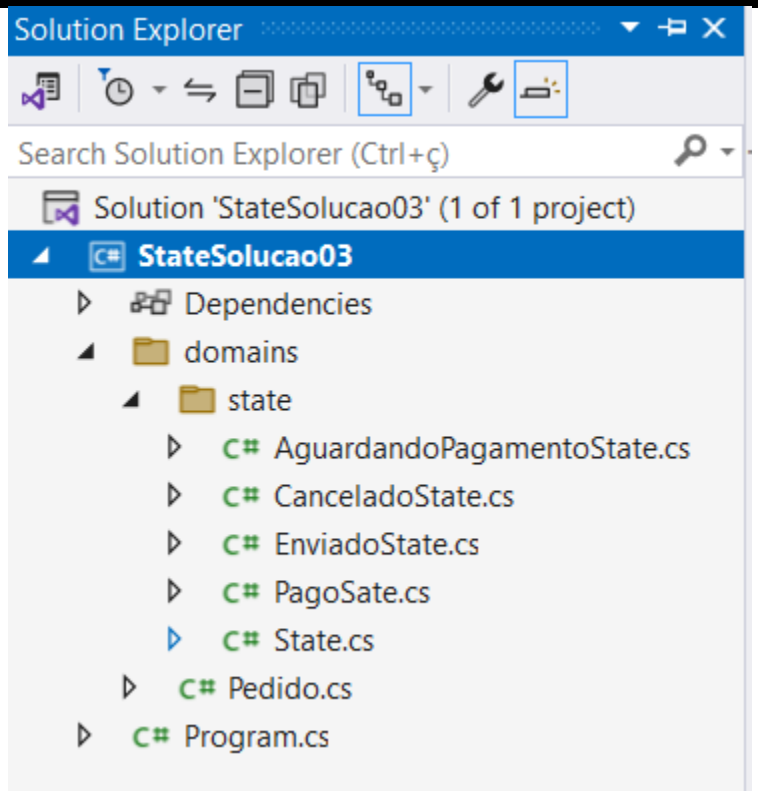


Implementação State – C# Solução 03

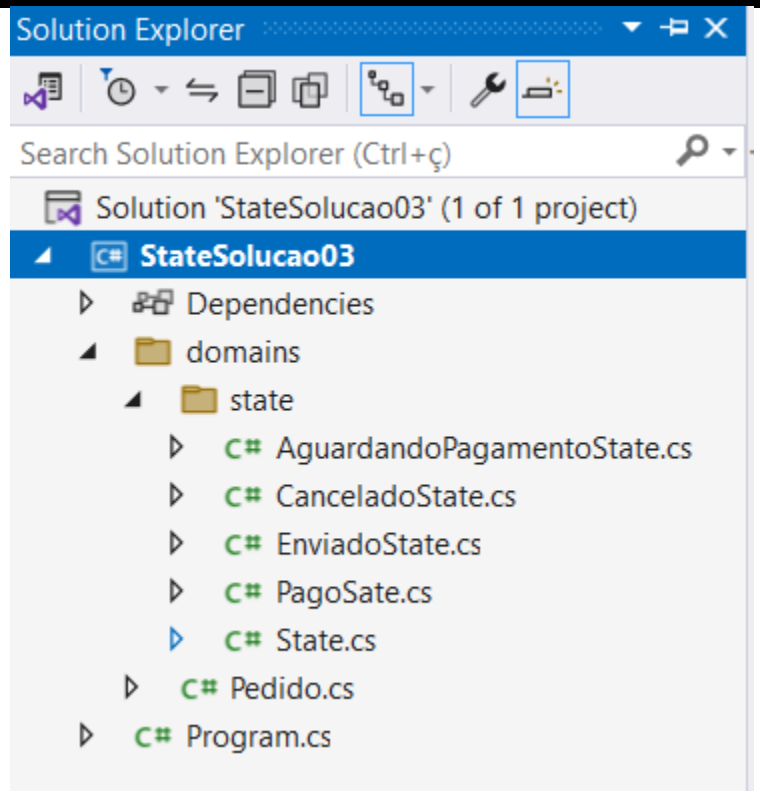
Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

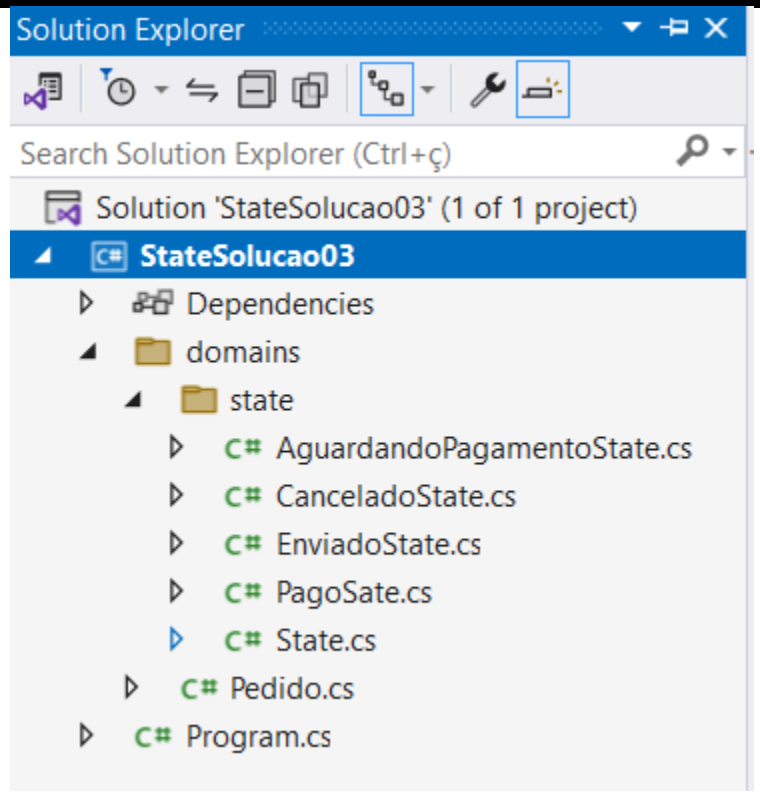
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace StateSolucao03.domains.state
8  {
       17 references
9      public interface State
10     {
           5 references
11         public void sucessoAoPagar();
           5 references
12         public void cancelarPedido();
           5 references
13         public void despacharPedido();
14
15     }
16 }
```



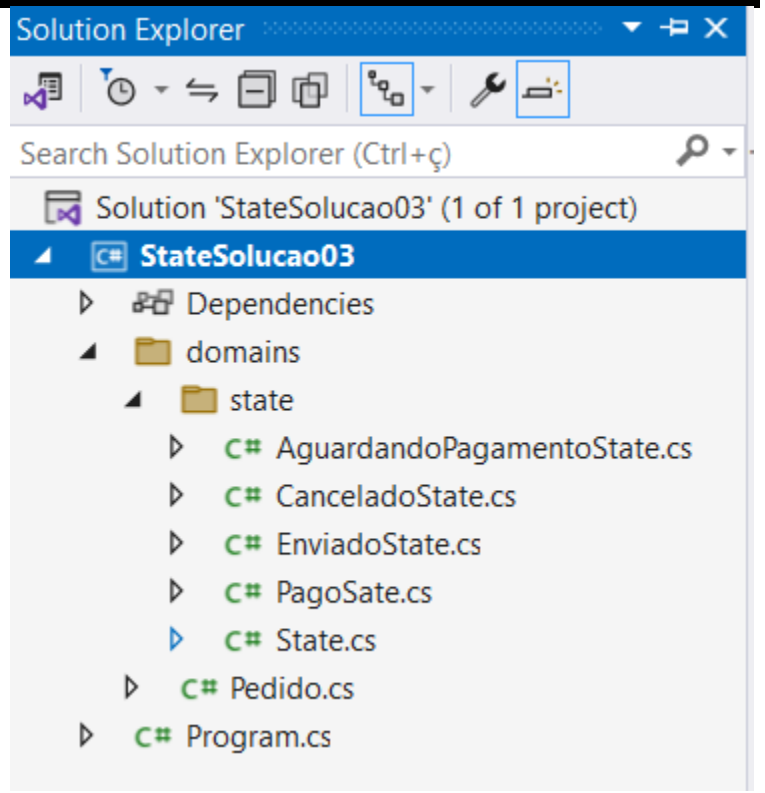
```
1   using ...
8
9   namespace StateSolucao03.domains
10  {
        13 references
11      public class Pedido
12      {
            8 references
13          public State estadoAtual { get; set; }
            2 references
14          public Pedido() {
15              Console.WriteLine("Pedido Aguardando Pagamento");
16              //define estado atual
17              this.estadoAtual = new AguardandoPagamentoState(this);
18          }
            2 references
19          public void sucessoAoPagar()
20          {
21              try {
22                  Console.WriteLine("Pedido Pago");
23                  this.estadoAtual.sucessoAoPagar();
24              } catch (Exception ex) {
25                  Console.WriteLine(ex.Message);
26              }
27          }
            1 reference
28          public void cancelarPedido()
29          {
30              try {
31                  Console.WriteLine("Pedido Cancelado");
32                  this.estadoAtual.cancelarPedido();
33              }
34              catch (Exception ex) {
35                  Console.WriteLine(ex.Message);
36              }
37          }
```



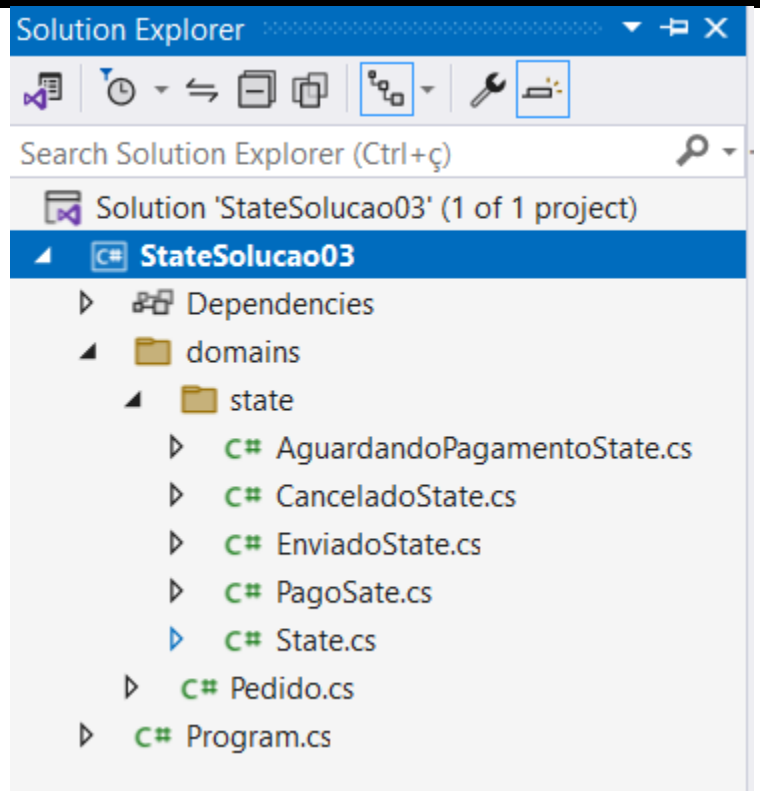
```
38          public void despacharPedido()
39          {
40              try {
41                  Console.WriteLine("Pedido enviado");
42                  this.estadoAtual.despacharPedido();
43              }
44              catch (Exception ex) {
45                  Console.WriteLine(ex.Message);
46              }
47          }
48
49      }
50  }
51
```



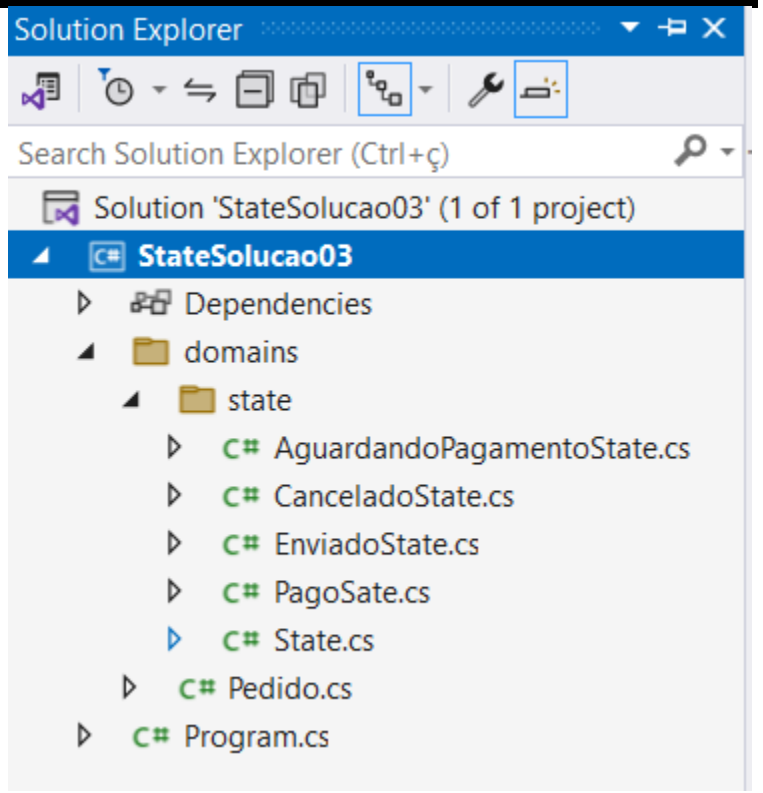
```
using StateSolucao03.domains;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StateSolucao03.domains.state
{
    2 referências
    internal class AguardandoPagamentoState : State
    {
        private Pedido pedido;
        1 referência
        public AguardandoPagamentoState(Pedido pedido)
        {
            this.pedido = pedido;
        }
        2 referências
        void State.cancelarPedido()
        {
            this.pedido.estadoAtual = new CanceladoState(pedido);
        }
        2 referências
        void State.despacharPedido()
        {
            throw new Exception("Operacao não suportada, " +
                "pedido ainda não foi pago");
        }
        2 referências
        void State.sucessoAoPagar()
        {
            this.pedido.estadoAtual = new PagoSate(pedido);
        }
    }
}
```



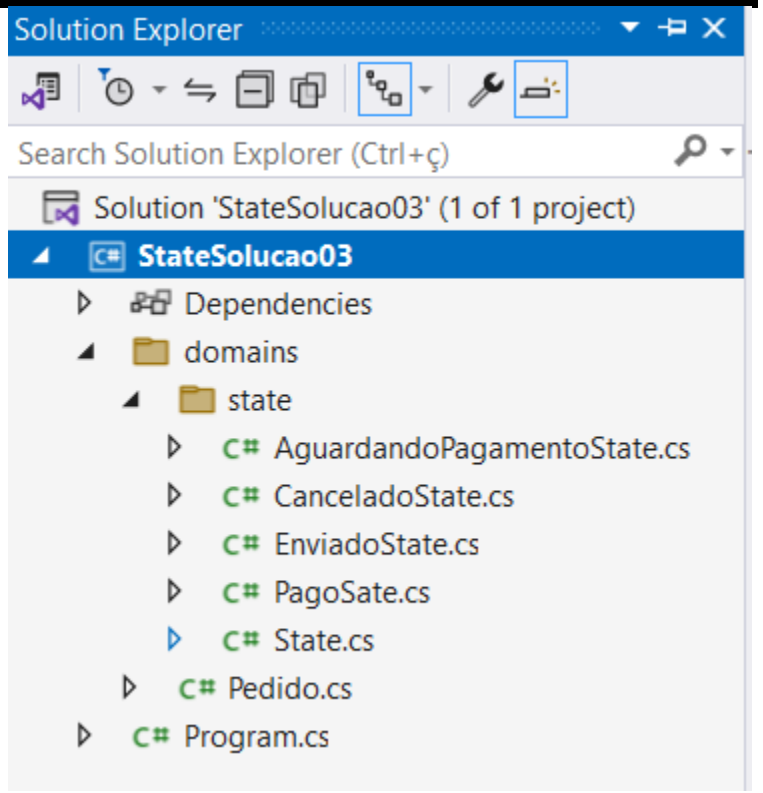
```
1   using StateSolucao03.domains;
2   using System;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace StateSolucao03.domains.state
9   {
        3 referências
10      public class CanceladoState : State
11      {
12          private Pedido pedido;
13
            2 referências
14          public CanceladoState(Pedido pedido)
15          {
16              this.pedido = pedido;
17          }
18
            2 referências
19          void State.cancelarPedido()
20          {
21              throw new Exception("Operacao não suportada, pedido foi cancelado");
22          }
23
            2 referências
24          void State.despacharPedido()
25          {
26              throw new Exception("Operacao não suportada, pedido foi cancelado");
27          }
28
            2 referências
29          void State.sucessoAoPagar()
30          {
31              throw new Exception("Operacao não suportada, pedido foi cancelado");
32          }
33      }
34  }
```



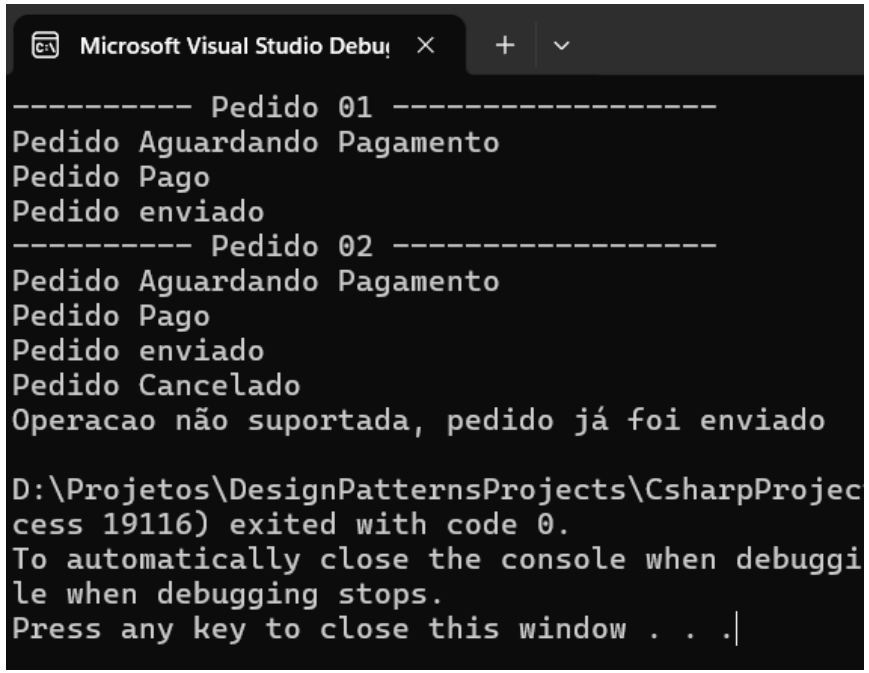
```
using StateSolucao03.domains;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StateSolucao03.domains.state
{
    2 referências
    public class EnviadoState : State
    {
        private Pedido pedido;
        1 referência
        public EnviadoState(Pedido pedido)
        {
            this.pedido = pedido;
        }
        2 referências
        void State.cancelarPedido()
        {
            throw new Exception("Operacao não suportada, pedido já foi enviado");
        }
        2 referências
        void State.despacharPedido()
        {
            throw new Exception("Operacao não suportada, pedido já foi enviado");
        }
        2 referências
        void State.sucessoAoPagar()
        {
            throw new Exception("Operacao não suportada, pedido já foi enviado");
        }
    }
}
```

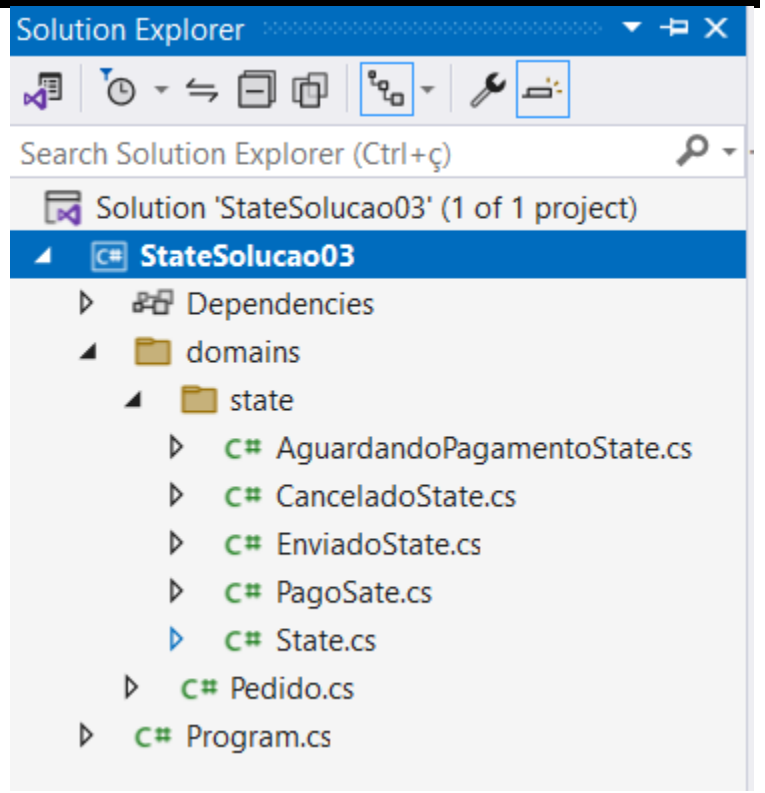


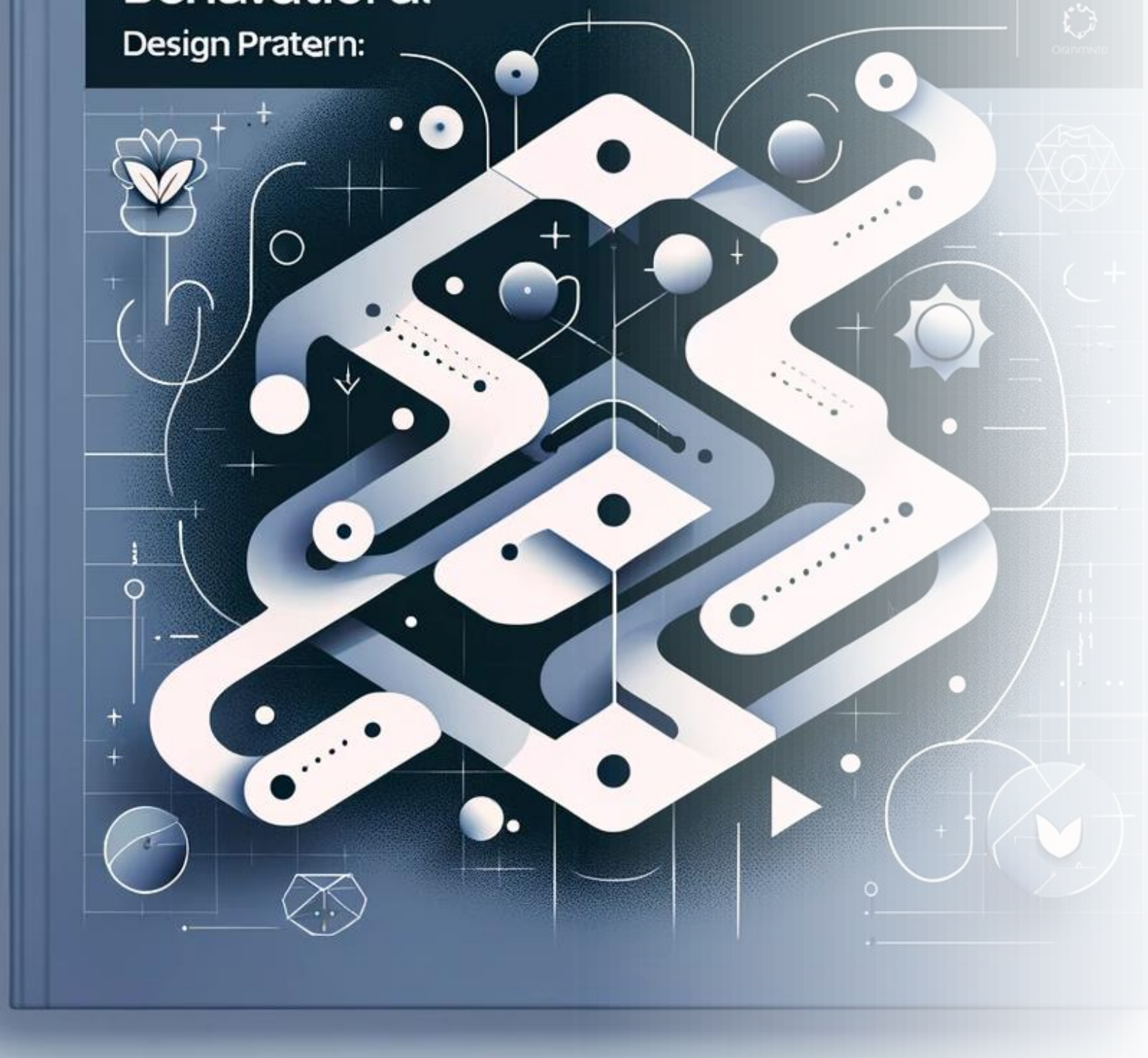
```using StateSolucao03.domains;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace StateSolucao03.domains.state
{
    2 referências
    public class PagoSate : State
    {
        private Pedido pedido;
        1 referência
        public PagoSate(Pedido pedido)
        {
            this.pedido = pedido;
        }
        2 referências
        void State.cancelarPedido()
        {
            this.pedido.estadoAtual = new CanceladoState(pedido);
        }
        2 referências
        void State.despacharPedido()
        {
            this.pedido.estadoAtual = new EnviadoState(pedido);
        }
        2 referências
        void State.sucessoAoPagar()
        {
            throw new Exception("Operacao não suportada, pedido já foi pago");
        }
    }
}
```



```
using StateSolucao03.domains;

try
{
    Console.WriteLine("---------- Pedido 01 ------------------");
    //Faça seus teste aqui!!!
    Pedido pedido = new Pedido();
    pedido.sucessoAoPagar();
    pedido.despacharPedido();

    Console.WriteLine("---------- Pedido 02 ------------------");
    Pedido pedido2 = new Pedido();
    pedido2.sucessoAoPagar();
    pedido2.despacharPedido();
    pedido2.cancelarPedido();


}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```





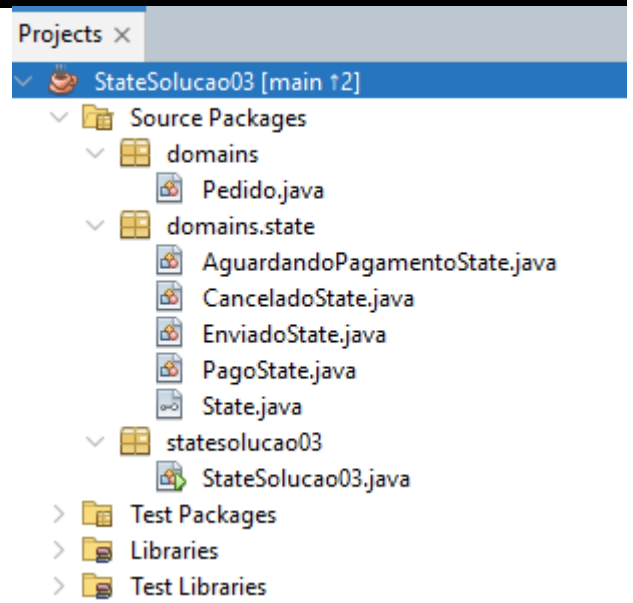


Implementação State – Java Solução 03

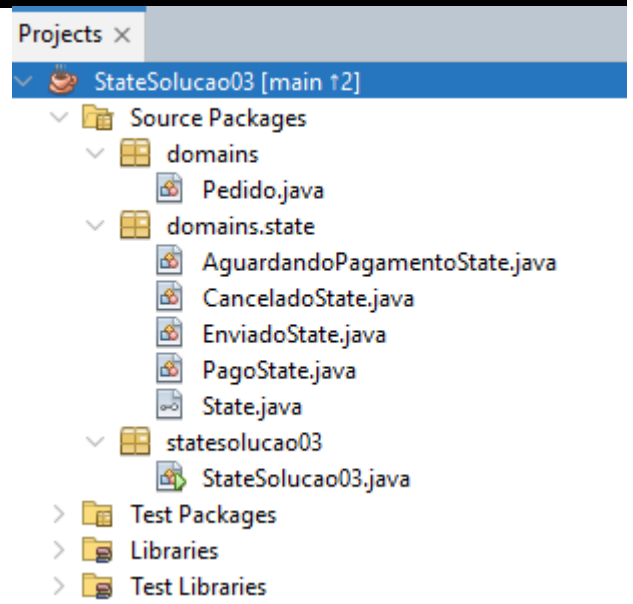
Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

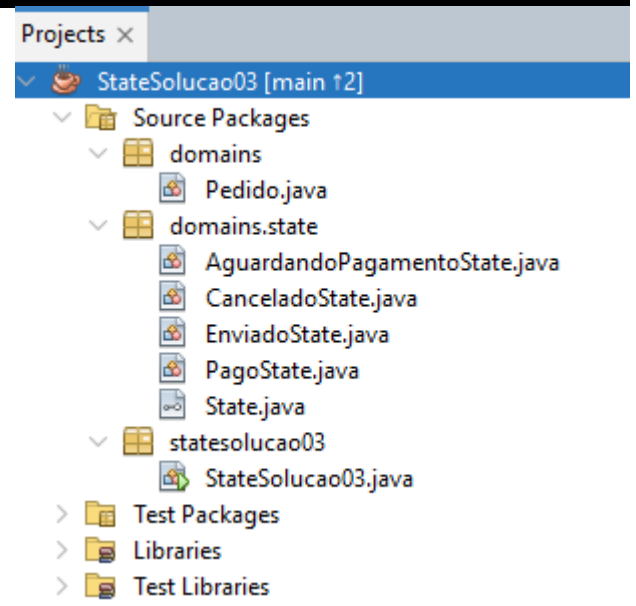
```
1  ...4 lines
5  package domains.state;
6
7  /**...4 lines */
   public interface State {
12
       public void sucessoAoPagar();
       public void cancelarPedido();
       public void despacharPedido();
16
17 }
```



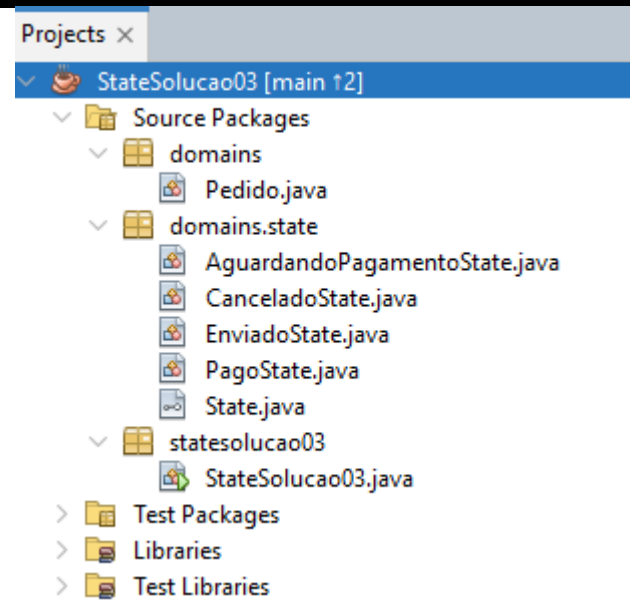
```java
...4 lines
package domains;

import domains.state.AguardandoPagamentoState;
import domains.state.State;

/**...4 lines */
public class Pedido {

    //estado atual do pedido.
    private State estadoAtual;

    public Pedido() {
        System.out.println("Pedido aguardando pagamento");
        //define o estado atual
        this.estadoAtual = new AguardandoPagamentoState(this);
    }
    public void sucessoAoPagar(){
        try{
            System.out.println("Pedido Pago");
            this.estadoAtual.sucessoAoPagar();
        } catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
    public void cancelarPedido(){
        try{
            System.out.println("Pedido Cancelar");
            this.estadoAtual.cancelarPedido();
        } catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
    public void despacharPedido(){
        try{
            System.out.println("Pedido Enviado");
            this.estadoAtual.despacharPedido();
        } catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
    public void setEstadoAtual(State estadoAtual) {
        this.estadoAtual = estadoAtual;
    }
}
```



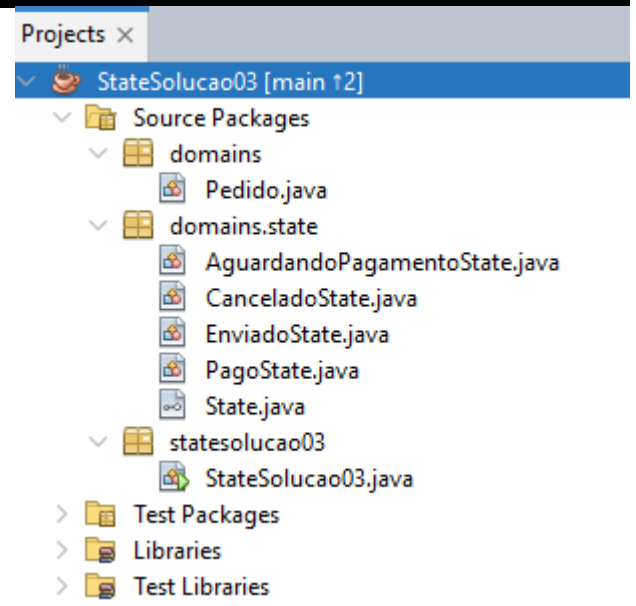
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class AguardandoPagamentoState implements State{

    private Pedido pedido;

    public AguardandoPagamentoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        this.pedido.setEstadoAtual(new PagoState(pedido));
    }

    @Override
    public void cancelarPedido() {
        this.pedido.setEstadoAtual(new CanceladoState(pedido));
    }

    @Override
    public void despacharPedido() {
        try {
            throw new Exception("Operação não suportada - pedido ainda não foi pago");
        } catch (Exception ex) {
            Logger.getLogger(AguardandoPagamentoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



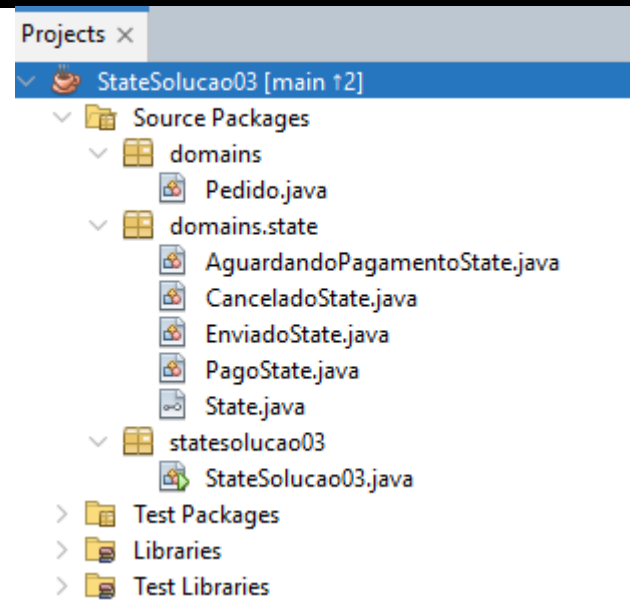
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class CanceladoState implements State{

    private Pedido pedido;

    public CanceladoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void cancelarPedido() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void despacharPedido() {
        try {
            throw new Exception("Operação não suportada - pedido cancelado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



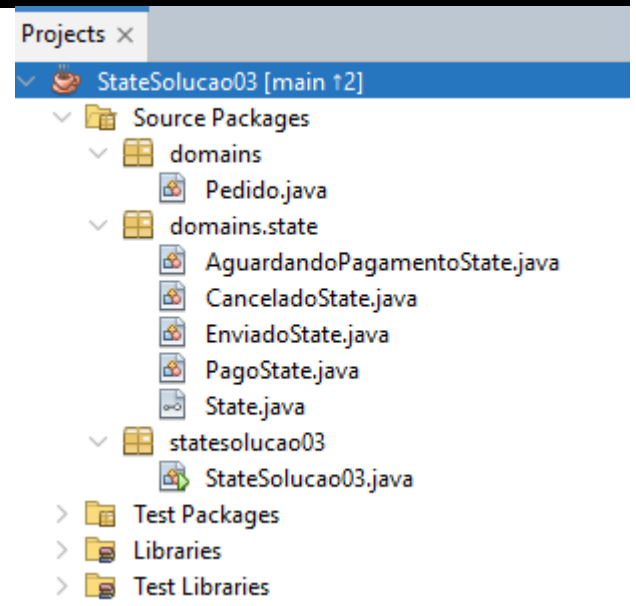
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author jeffe
 */
public class EnviadoState implements State{

    private Pedido pedido;

    public EnviadoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        try {
            throw new Exception("Operação não suportada - pedido enviado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void cancelarPedido() {
        try {
            throw new Exception("Operação não suportada - pedido enviado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void despacharPedido() {
        try {
            throw new Exception("Operação não suportada - pedido enviado");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



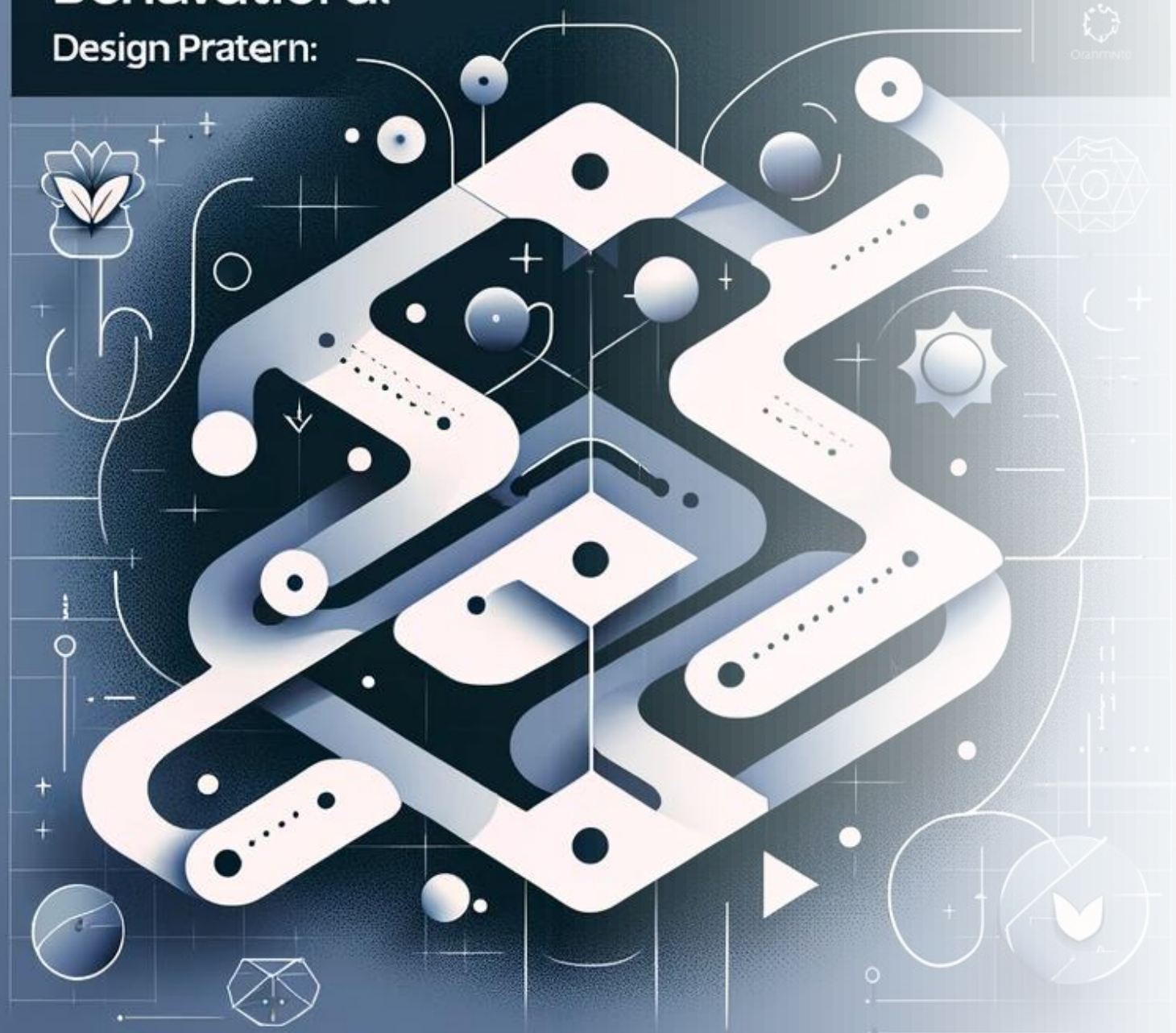
```java
...4 lines
package domains.state;

import domains.Pedido;
import java.util.logging.Level;
import java.util.logging.Logger;

/**...4 lines */
public class PagoState implements State{

    private Pedido pedido;

    public PagoState(Pedido pedido) {
        this.pedido = pedido;
    }

    @Override
    public void sucessoAoPagar() {
        try {
            throw new Exception("Operação não suportada - pedido já foi pago");
        } catch (Exception ex) {
            Logger.getLogger(CanceladoState.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    @Override
    public void cancelarPedido() {
        this.pedido.setEstadoAtual(new CanceladoState(pedido));
    }

    @Override
    public void despacharPedido() {
        this.pedido.setEstadoAtual(new EnviadoState(pedido));
    }

}
```



```java
...4 lines
package statesolucao03;

import domains.Pedido;

/**...4 lines */
public class StateSolucao03 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        try
        {
            System.out.println("---------- Pedido 01 ------------------");
            //Faça seus teste aqui!!!
            Pedido pedido = new Pedido();
            pedido.sucessoAoPagar();
            pedido.despacharPedido();

            System.out.println("---------- Pedido 02 ------------------");
            Pedido pedido2 = new Pedido();
            pedido2.sucessoAoPagar();
            pedido2.despacharPedido();
            pedido2.cancelarPedido();


        } catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }

}
```




```
run:
---------- Pedido 01 ------------------
Pedido aguardando pagamento
Pedido Pago
Pedido Enviado
---------- Pedido 02 ------------------
Pedido aguardando pagamento
Pedido Pago
Pedido Enviado
Pedido Cancelar
mar. 21, 2024 4:34:32 PM domains.state.EnviadoState cancelarPedido
GRAVE: null
java.lang.Exception: Operação não suportada - pedido cancelado
        at domains.state.EnviadoState.cancelarPedido(EnviadoState.java:35)
        at domains.Pedido.cancelarPedido(Pedido.java:37)
        at statesolucao03.StateSolucao03.main(StateSolucao03.java:31)

BUILD SUCCESSFUL (total time: 0 seconds)
```



Implementação State Prós e Contras

Padrões de Projeto Comportamentais I

Prof. Me Jefferson Passerini

- **Prós e Contras**
- **Solução 02 (com estados no construtor da classe Pedido)**
  - Ao olhar o código da classe Pedido é possível identificar de forma explícita todos os estados que ela pode assumir.
  - As dependências ficam centralizadas na classe Pedido e as classes de estado não precisam conhecer quais são os outros estados.
  - Ao adicionar um novo estado no sistema, a classe Pedido terá que ser alterada para suportar o novo estado (criação de novo atributo e seu respectivo método getter).
  - Mesmo tendo que alterar a classe Pedido, a lógica dos novos estados ficarão fora dela, deste modo a classe Pedido não crescerá de forma desordenada em complexidade.
  - Essa abordagem não respeita o Open/Closed Principle em sua totalidade.

- **Prós e Contras**
- **Solução 03 (sem estados no construtor da classe Pedido)**
  - Ao olhar o código da classe Pedido não é possível identificar de forma explícita todos os estados que ela pode assumir.
  - As dependências ficam dispersas entre as classes de estado. Cada estado precisa conhecer as outras classes de estado para quais eles devem transitar.
  - Ao adicionar um novo estado no sistema, a classe Pedido não precisará ser alterada. Segue totalmente o Open/Closed Principle.