

Padrões de Projeto

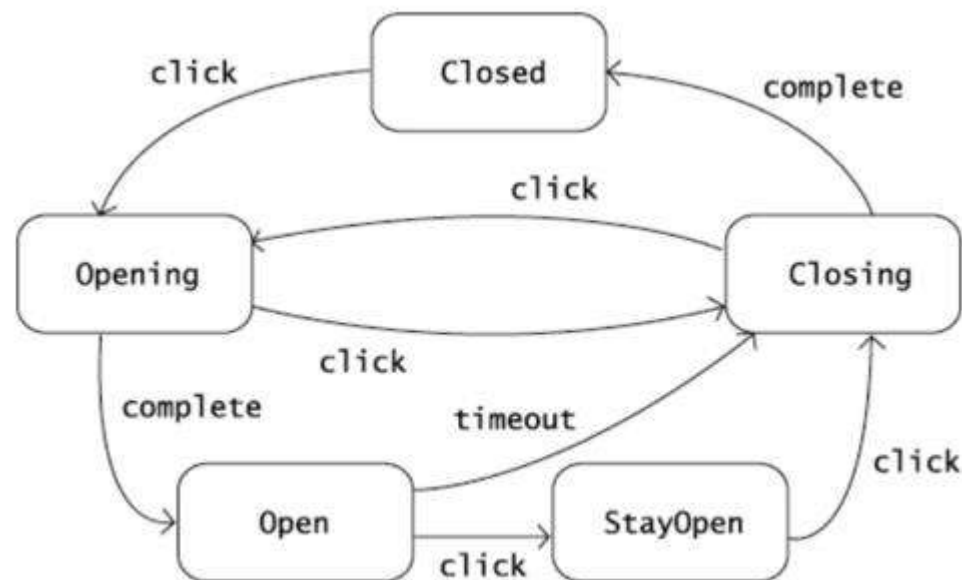
Prof. Adilson Vahldick

Departamento de Engenharia de Software

Udesc Ibirama

Objetivos da aula

- Conhecer e aplicar o padrão
 - State



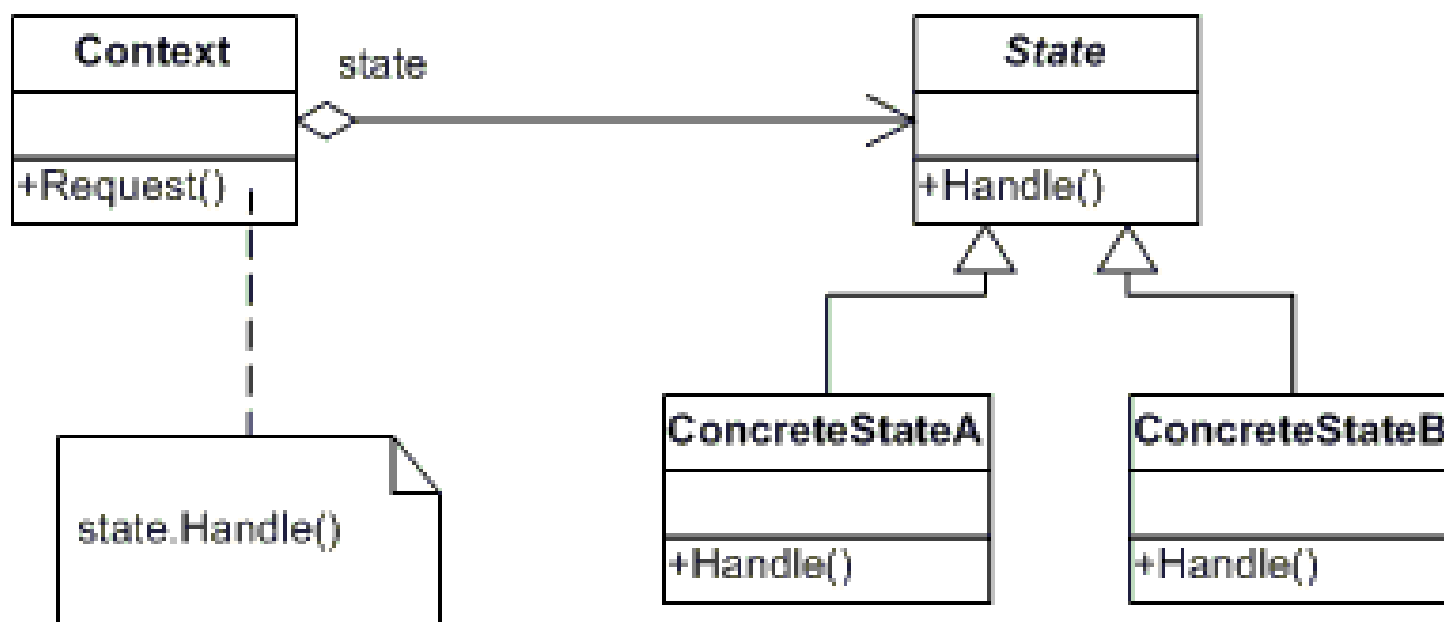
Problema (1)

Ventilador
- estadoAtual: int
+ pressionarBotao():void

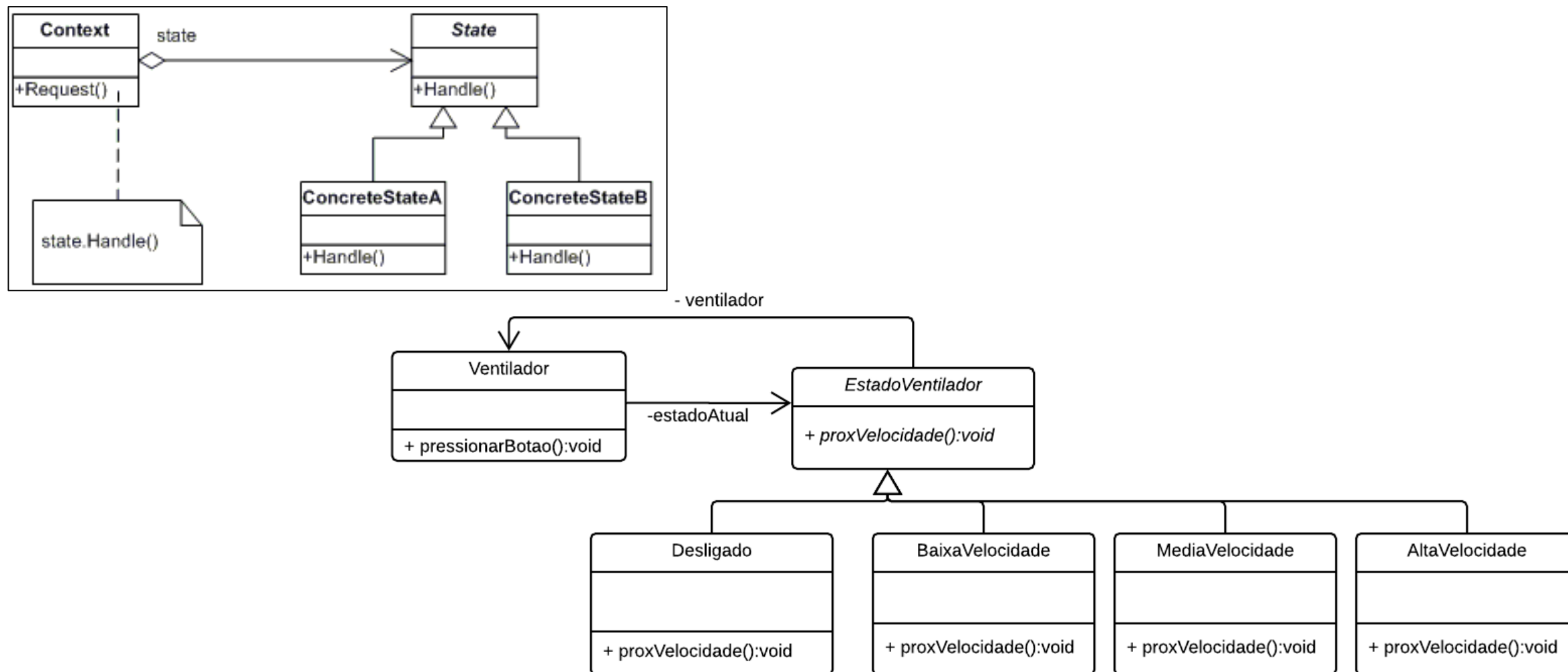
```
public class Ventilador {  
  
    private int estadoAtual = 0;  
  
    public void pressionarBotao() {  
        if (estadoAtual == 0) {  
            estadoAtual = 1;  
            System.out.println("Baixa velocidade");  
        } else {  
            if (estadoAtual == 1) {  
                estadoAtual = 2;  
                System.out.println("Media velocidade");  
            } else {  
                if (estadoAtual == 2) {  
                    estadoAtual = 3;  
                    System.out.println("Alta velocidade");  
                } else {  
                    estadoAtual = 0;  
                    System.out.println("Desligado");  
                }  
            }  
        }  
    }  
}
```

Solução (1)

- **State:** Permite a um objeto alterar seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado sua classe.



Solução (2)

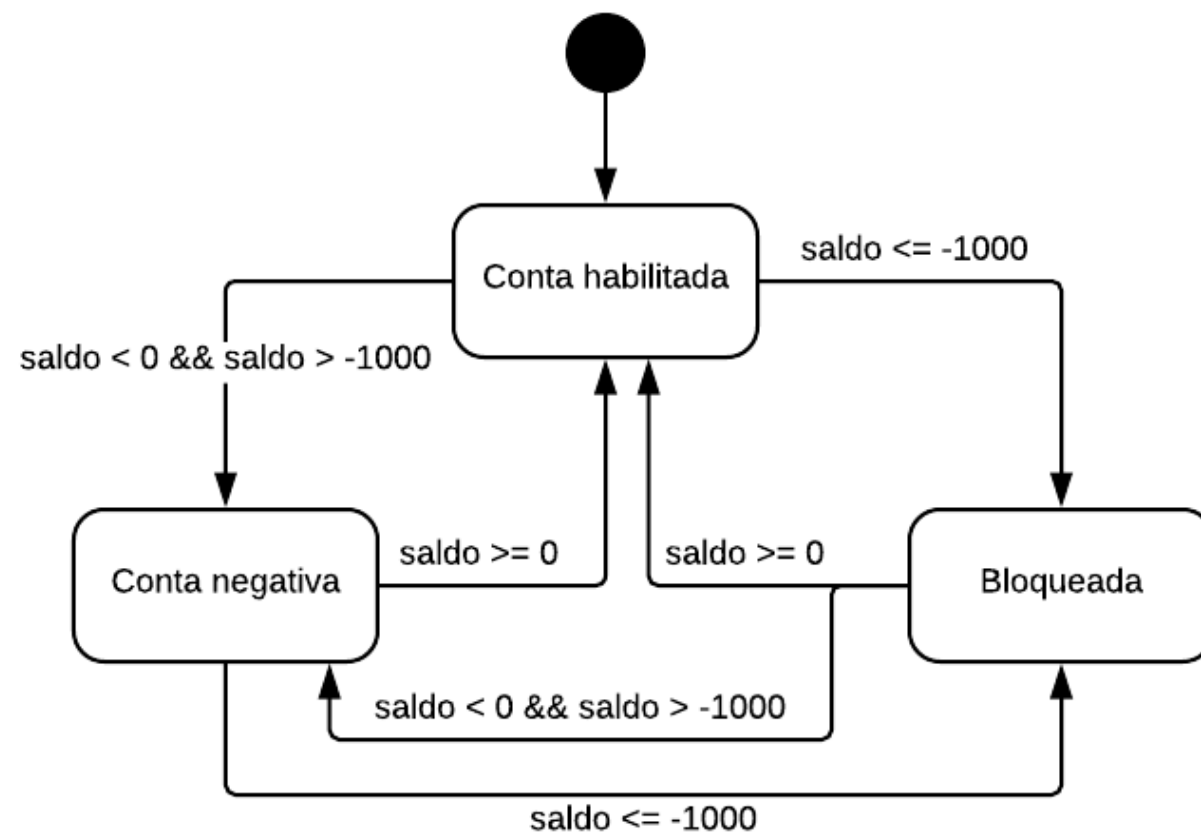
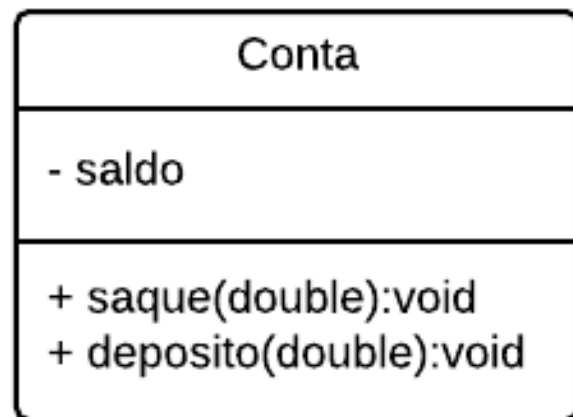


Solução (3)

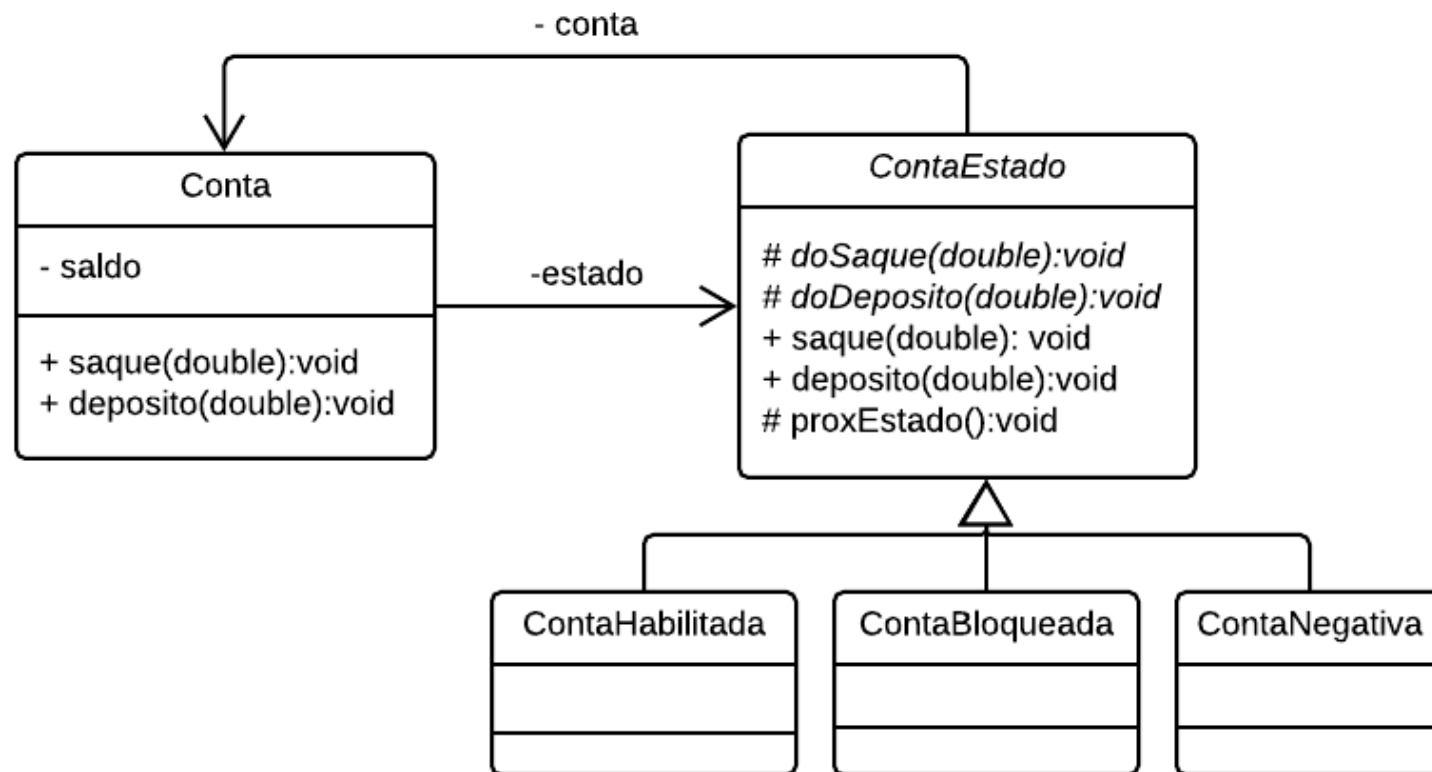
```
public class Ventilador {  
  
    private EstadoVentilador estadoAtual;  
  
    public Ventilador() {  
        this.estadoAtual = new Desligado(this);  
    }  
  
    public void setEstadoAtual(EstadoVentilador estadoAtual) {  
        this.estadoAtual = estadoAtual;  
    }  
  
    public void pressionarBotao() {  
        this.estadoAtual.proxVelocidade();  
    }  
}
```

```
public class Desligado extends EstadoVentilador {  
  
    public Desligado(Ventilador ventilador) {  
        super(ventilador);  
        System.out.println("Desligado");  
    }  
  
    public void proxVelocidade() {  
        ventilador.setEstadoAtual(new BaixaVelocidade(ventilador));  
    }  
}
```

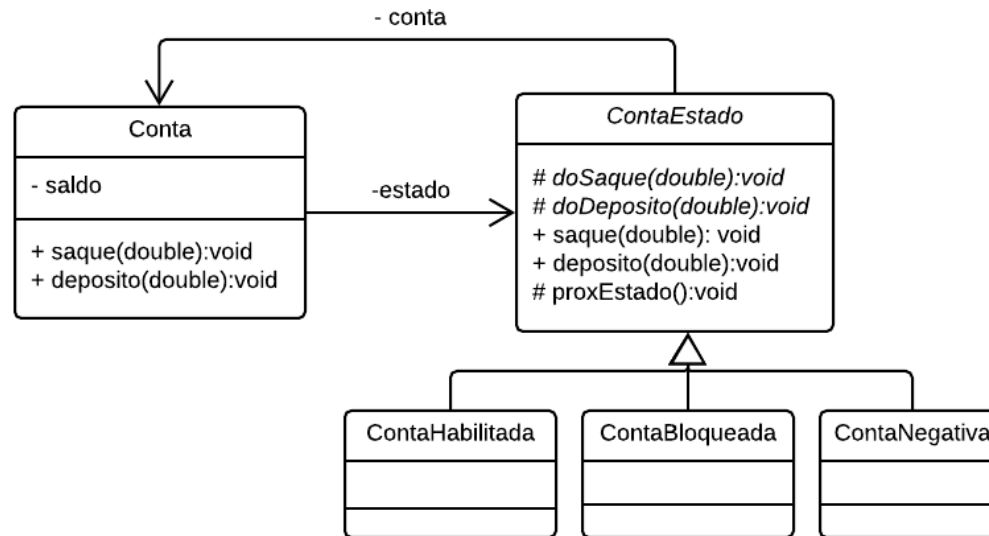
Problema 2



Solução 2 (1)



Solução 2 (2)



```
public abstract class ContaEstado {

    protected Conta conta;

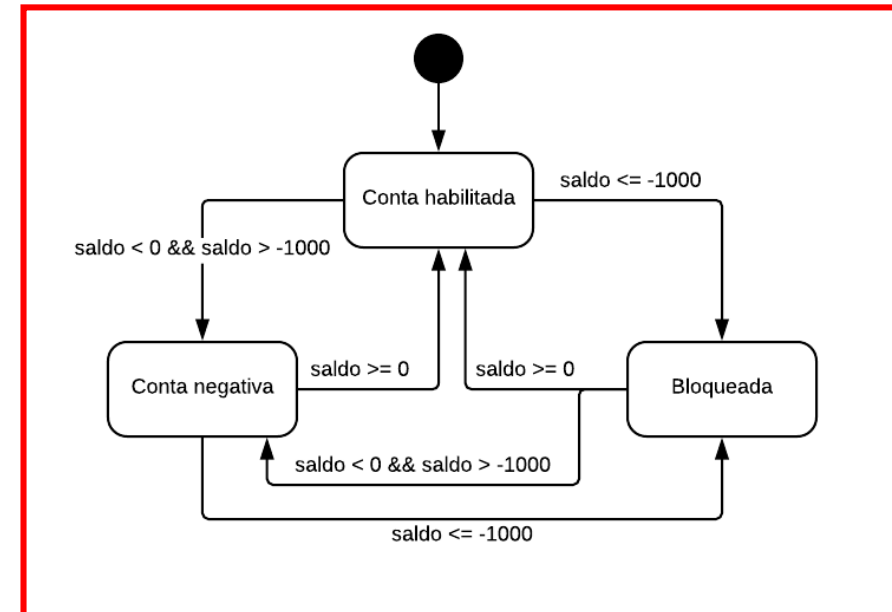
    public ContaEstado(Conta conta) {
        this.conta = conta;
    }

    public final void deposito(double valor) throws Exception {
        doDeposito(valor);
        proxEstado();
    }

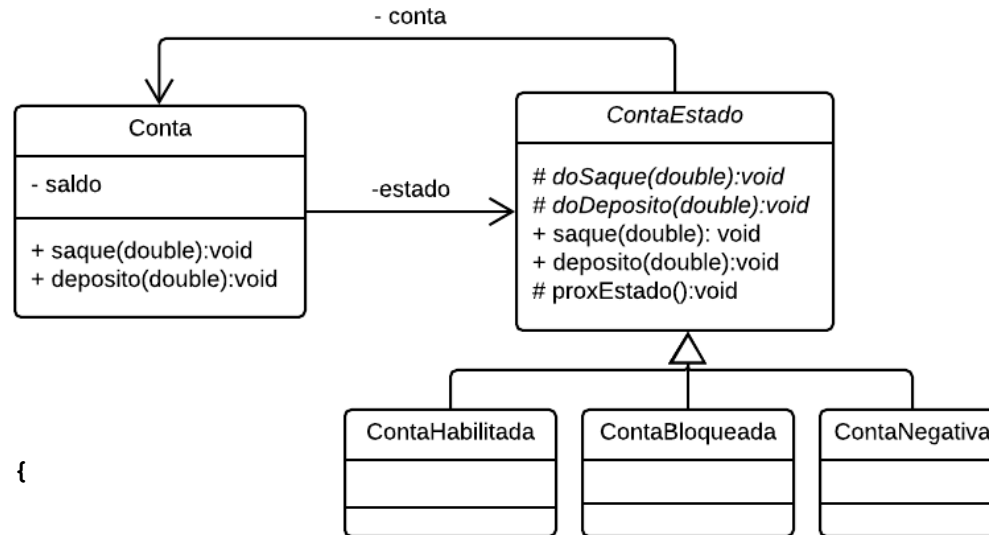
    public final void saque(double valor) throws Exception {
        doSaque(valor);
        proxEstado();
    }

    protected abstract void doDeposito(double valor) throws Exception;
    protected abstract void doSaque(double valor) throws Exception;

    protected void proxEstado() {
        ...
    }
}
```



Solução 2 (3)



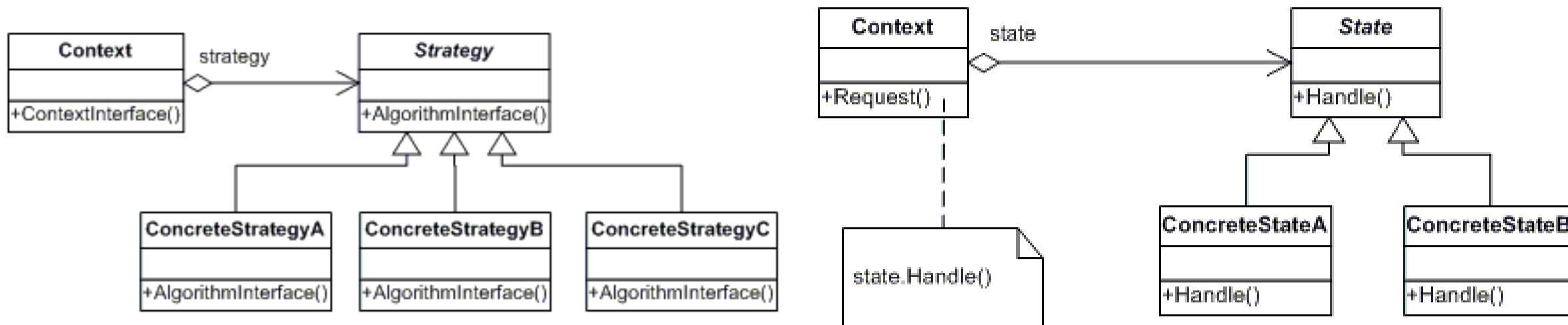
```
public class ContaBloqueada extends ContaEstado {
    ...

    protected void doSaque(double valor) throws Exception {
        throw new Exception("Conta bloqueada, n\u00E3o \u00E9 poss\u00EDvel fazer saque");
    }
}

public class ContaNegativa extends ContaEstado {
    ...

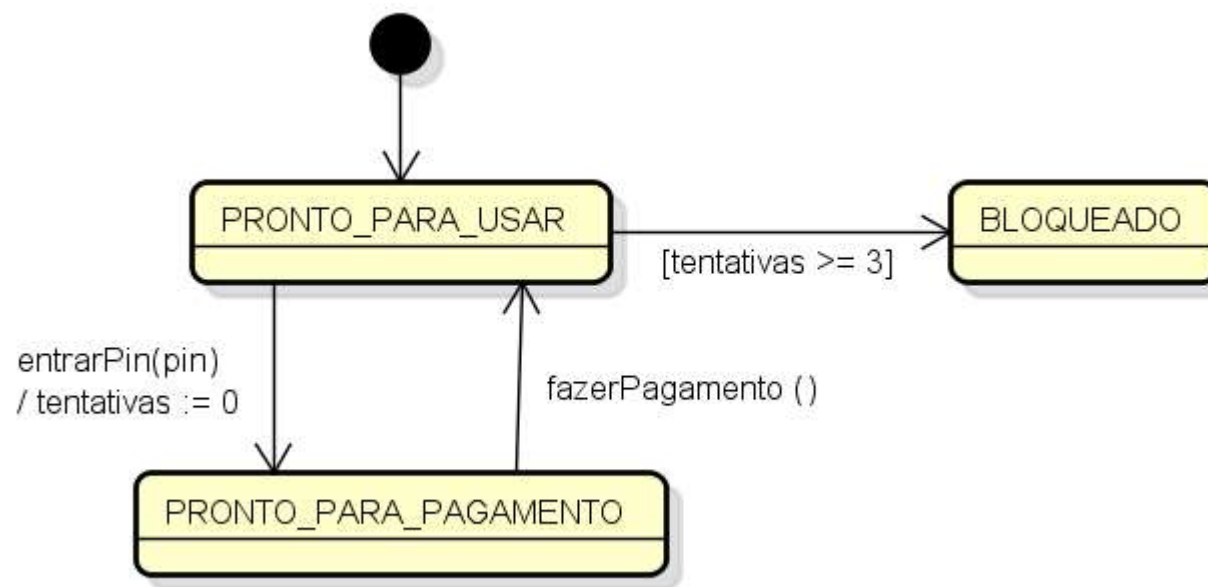
    protected void doSaque(double valor) throws Exception {
        this.conta.setSaldo(conta.getSaldo() + valor);
        // na verdade a mensagem abaixo deveria ser resolvida com Observer
        System.out.println("Conta negativa, aten\u00E7\u00E3o para o saldo");
    }
}
```

Diferença entre Strategy e State



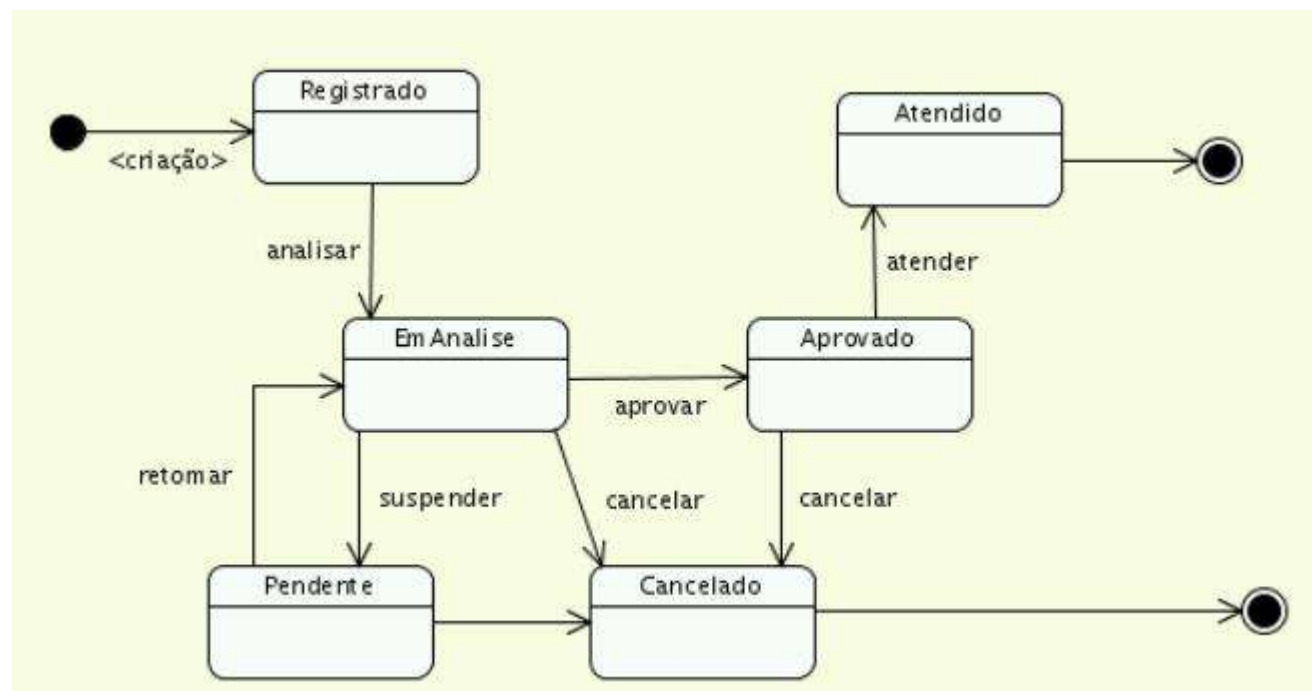
Exercício (1) – state3

- Abaixo está representado os estados de um Cartão bancário. Aplique o padrão *state* para refletir o diagrama. (Falta o estado Bloqueado)



Exercício (2) – state4

- O diagrama abaixo representa os estados de um pedido. O projeto possui a classe Pedido e a de cliente. Aplique o padrão *State*



Exercício (3)

- Desenvolva um caixa eletrônico seguindo o diagrama de estados abaixo.

