





## Alex Sander Resende de Deus

A 25 anos ensinando programação a jovens e adultos.

Apaixonado por tecnologia é atualmente coordenador de cursos na ETEC Albert Einstein. Na FIAP atua como professor da FIAP School, lecionando C#, SQLServer e Desenvolvimento Mobile

## AULA 10

# Classes e métodos concretos e abstratos

# Classes abstratas e concretas

- Uma classe abstrata é desenvolvida para representar entidades e conceitos abstratos.
- A classe abstrata é sempre uma superclasse que não possui instâncias.
- Ela define um modelo para uma funcionalidade e fornece uma implementação incompleta (a parte genérica dessa funcionalidade) que é compartilhada por um grupo de classes derivadas (subclasses).
- Cada uma das classes derivadas completa a funcionalidade da classe abstrata adicionando um comportamento específico.

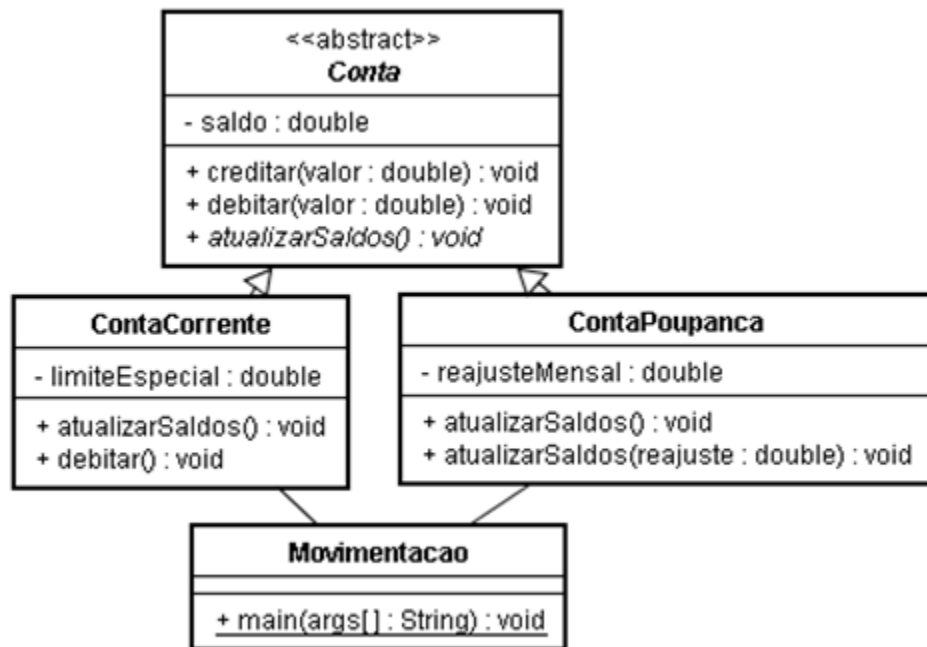
# Métodos abstratos

- Uma classe abstrata pode conter métodos concretos, porém, um método abstrato só pode ser definido em uma classe abstrata.
- Esses métodos são implementados nas suas subclasses (concretas) com o objetivo de definir um comportamento (regras) específico.
- Um método abstrato define apenas a assinatura do método e, portanto, não contém código.

# Métodos abstratos

- No exemplo do controle bancário, a classe Conta nunca será utilizada para instanciar um objeto.
- A finalidade da classe Conta é somente a generalização dos produtos bancários, portanto, conceitualmente ela deve ser definida como uma classe abstrata.
- Partindo desse princípio, o método atualizarSaldo também é um forte candidato a ser um método abstrato, porque, nesse exemplo, cada produto bancário tem sua própria regra de cálculo.

# Projeto: Controle Bancário



## Classe abstrata: Conta

|                 |  |
|-----------------|--|
| Método concreto | <b>creditar:</b> Recebe um valor por parâmetro e soma ao atributo saldo  |
| Método concreto | <b>debitar:</b> Recebe um valor por parâmetro e subtrai do atributo saldo desde que haja saldo suficiente, caso não tenha, apresenta mensagem de saldo insuficiente. |
| Método abstrato | <u>atualizarSaldo</u> : Somente a assinatura do método.  |

Classe: ContaCorrente (subclasse de Conta)

|         |   |
|---------|---|
| Métodos | <u>atualizarSaldo</u> : Implementação do método (abstrato na superclasse) que verifica se o atributo saldo <u>esta</u> negativo, caso esteja, calcula 8% (0.08) sobre o valor excedente e subtrai do saldo (Cobra juros pela utilização de limite especial).<br>Apresentar o saldo anterior e o saldo atualizado. |
|         | <b>debitar:</b> Sobrescrever o método debitar considerando o atributo <u>limiteEspecial</u> . O saldo poderá ficar negativo até o valor indicado em <u>limiteEspecial</u> .   |

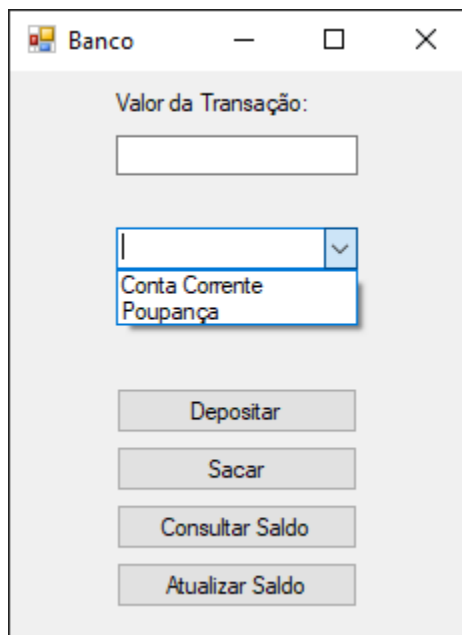
Classe: ContaPoupanca (subclasse de Conta)

|         |  |
|---------|--|
| Métodos | <u>atualizarSaldo</u> (sem parametro): Método (abstrato na superclasse) herdado que deve ter, pelo menos, a assinatura inserida na subclasse.  |
|         | <u>atualizarSaldo</u> (com parametro): Sobrecarregar o método <u>atualizarSaldo</u> de modo que ele receba por parâmetro uma porcentagem para reajuste (um valor decimal <u>double</u> ).<br>Calcular a porcentagem informada sobre o saldo e somar ao saldo (Rendimento da poupança).<br>Armazenar a porcentagem informada no atributo <u>reajusteMensal</u> .<br>Apresentar o saldo anterior e o saldo atualizado. |



| Formulário |  |
|------------|--|
|            | <ul style="list-style-type: none"><li>• Instanciar um objeto do tipo <u>ContaCorrente</u> chamado cc1 com saldo inicial de 500 e limite especial de 1000.</li></ul>  |
|            | <ul style="list-style-type: none"><li>• Instanciar um objeto do tipo <u>ContaPoupanca</u> chamado cp1 com saldo inicial de 5000 e <u>reajusteMensal</u> de 1% (0.01)</li></ul>   |
|            | <ul style="list-style-type: none"><li>• Apresentar um menu com as opções:<ul style="list-style-type: none"><li>1 – Conta corrente</li><li>2 – <u>Poupanca</u></li><li>0 – Sair</li></ul></li></ul>   |
|            | <ul style="list-style-type: none"><li>• Apresentar outro menu com as opções:<ul style="list-style-type: none"><li>1 – Depositar</li><li>2 – Sacar</li><li>3 – Consultar saldo</li><li>4 – Reajustar saldo</li><li>0 – Sair</li></ul></li></ul> |
|            | <ul style="list-style-type: none"><li>• Realizar as chamadas aos métodos de acordo com as opções do usuário.</li></ul>   |
|            | <b>Obs.:</b> No reajuste da poupança informar a porcentagem a ser aplicada (em formato decimal)  |

# Formulário



Banco

Valor da Transação:

▼  
Conta Corrente  
Poupança

Depositar

Sacar

Consultar Saldo

Atualizar Saldo

# SuperClasse Conta

```
abstract class Conta
{
    public double saldo { get; set; }

    public Conta()
    {
        saldo = 0;
    }

    public Conta(double saldo)
    {
        this.saldo = saldo;
    }

    public virtual void debitar(double valor)
    {
        this.saldo -= valor;
        MessageBox.Show("Saque efetuado");
    }

    public void creditar(double valor)
    {
        this.saldo += valor;
        MessageBox.Show("Depósito Efetuado");
    }

    public abstract void atualizarSaldos(); // MÉTODO ABSTRATO, SOMENTE CONTÉM ASSINATURA
}
```

# Classe ContaCorrente

```
class ContaCorrente:Conta
{
    public double limiteEspecial { get; set; }

    public ContaCorrente()
    {
        this.limiteEspecial = 0;
        this.saldo = 0;
    }

    public ContaCorrente(double saldo,double limiteEspecial)
    {
        this.limiteEspecial = limiteEspecial;
        this.saldo = saldo;
    }

    public override void debitar(double valor)
    {
        if (valor <= this.saldo + this.limiteEspecial)
        {
            saldo -= valor;
            MessageBox.Show("Saque efetuado");
        }
        else
        {
            MessageBox.Show("Saldo Insuficiente");
        }
    }
}
```

```
public override void atualizarSaldos()
{
    double saldoAnterior = this.saldo;
    if (this.saldo < 0)
    {
        saldo += saldo * 0.08;
    }
    MessageBox.Show("Saldo Anterior: " + saldoAnterior.ToString() +
        "\nSaldo Atual: " + this.saldo);
}
```

# Classe ContaPoupança

```
class ContaPoupanca:Conta
{
    public double reajusteMensal { get; set; }

    public ContaPoupanca()
    {
        this.reajusteMensal = 0;
        this.saldo = 0;
    }

    public ContaPoupanca(double saldo,double reajusteMensal)
    {
        this.saldo = saldo;
        this.reajusteMensal = reajusteMensal;
    }

    public void atualizarSaldos(double reajuste)
    {
        double saldoAnterior = this.saldo;
        saldo += saldo * (reajuste / 100);

        MessageBox.Show("Saldo Anterior: " + saldoAnterior.ToString() +
            "\nSaldo Atual: " + this.saldo);
    }

    public override void atualizarSaldos()
    {
    }
}
```

# Código dos botões

```
ContaCorrente cc1=new ContaCorrente(500,1000);
ContaPoupanca cp1 = new ContaPoupanca(5000, 0.01);

private void btnDepositar_Click(object sender, EventArgs e)
{
    double valor = Convert.ToDouble(txtValor.Text);
    switch (cmbTipoConta.SelectedIndex)
    {
        case 0:
            cc1.creditar(valor);
            break;
        case 1:
            cp1.creditar(valor);
            break;
    }
}
```

```
private void btnSacar_Click(object sender, EventArgs e)
{
    double valor = Convert.ToDouble(txtValor.Text);
    switch (cmbTipoConta.SelectedIndex)
    {
        case 0:
            cc1.debitar(valor);
            break;
        case 1:
            if (valor <= cp1.saldo)
            {
                cp1.debitar(valor);
            }
            else
            {
                MessageBox.Show("Saldo insuficiente");
            }
            break;
    }
}

private void btnConsultarSaldo_Click(object sender, EventArgs e)
{
    switch (cmbTipoConta.SelectedIndex)
    {
        case 0:
            MessageBox.Show("Saldo em CC: R$" + cc1.saldo.ToString());
            break;
        case 1:
            MessageBox.Show("Saldo em Poupança: R$" + cp1.saldo.ToString());
            break;
    }
}
```

# Código dos botões

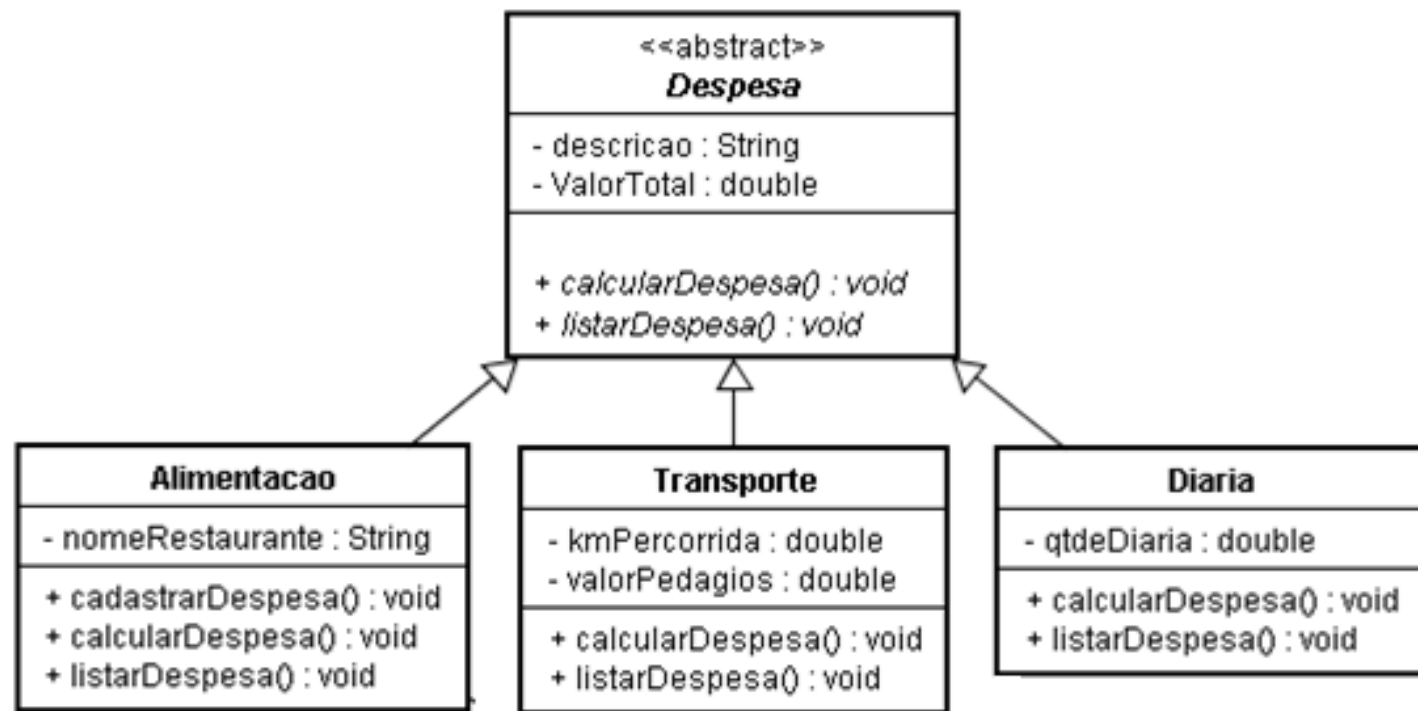
```
private void btnAtualizaSaldo_Click(object sender, EventArgs e)
{
    switch (cmbTipoConta.SelectedIndex)
    {
        case 0:
            cc1.atualizarSaldos();
            break;
        case 1:
            double reajuste = Convert.ToDouble(Interaction.InputBox("Digite a taxa de reajuste"));
            cp1.atualizarSaldos(reajuste);
            break;
    }
}
```



# Momento Hands On







Classe abstrata: Funcionario

|         |  |
|---------|--|
| Métodos | <u>cadastrarFuncionario</u> : Lê o nome, CPF e salário base e armazena em seus respectivos atributos. Inicializa com zero os atributo <u>salarioFinal</u> e <u>valorBonificacao</u>                |
|         | <u>calcularBonificacao</u> : Calcula 5% do salário final do funcionário e insere no atributo <u>valorBonificacao</u> através do métodos set.<br>Cálculo: <u>bonificacao</u> = salário final * 0.05 |
|         | <u>listarFuncionario</u> : Método abstrato   |
|         | <u>calcularSalarioFinal</u> : Método abstrato  |
|         | <u>apresentarSalario</u> : Método abstrato   |

Subclasse: Gerente

|         |   |
|---------|---|
| Métodos | <u>calcularSalarioFinal</u> : Lê o valor da gratificação e armazena no atributo <u>gratificacao</u> . Soma o atributo gratificação, o salário base e a bonificação e armazena no atributo <u>salarioFinal</u> . |
|         | <u>listarFuncionario</u> : (Implementação de método abstrato) Apresenta todos os atributos do objeto  |
|         | <u>apresentarSalario</u> : Apresentar o nome do funcionário, o valor do salário base, o valor da bonificação, o valor da gratificação e o valor do salário final.   |

## Subclasse: Vendedor

## Métodos

calcularSalarioFinal: Lê o valor da comissão e armazena no atributo valorComissao.  
Soma o atributo valorComissao, o salário base e a bonificação e armazena no atributo salarioFinal.

listarFuncionario: (Implementação de método abstrato) Apresenta todos os atributos do objeto

apresentarSalario: Apresentar o nome do funcionário, o valor do salário base, o valor da bonificação, o valor da comissão e o valor do salário final.

## Subclasse: Atendente

## Métodos

calcularSalarioFinal: Lê o valor do adicional noturno e armazena no atributo adicionalNoturno.  
Soma o atributo adicionalNoturno, o salário base e a bonificação e armazena no atributo salarioFinal.

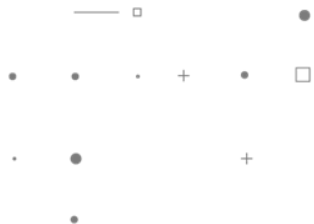
listarFuncionario: (Implementação de método abstrato) Apresenta todos os atributos do objeto

apresentarSalario: Apresentar o nome do funcionário, o valor do salário base, o valor da bonificação, o valor do adicional noturno e o valor do salário final.

| Formulário |   |
|------------|---|
|            | • Instanciar um objeto do tipo Gerente chamado gerente.   |
|            | • Instanciar um objeto do tipo Vendedor chamado vendedor.   |
|            | • Instanciar um objeto do tipo Atendente chamado atendente.   |
|            | • Apresentar um menu com as opções:<br>1 – Gerente<br>2 – Vendedor<br>3 – Atendente<br>0 – Sair   |
|            | • Apresentar um menu com as opções:<br>1 – Cadastrar <u>funcionario</u><br>2 – Calcular salário<br>3 – Calcular bonificação<br>4 – Apresentar salário final<br>0 – Sair |

# Interfaces

---



# Definição

- Uma **interface**, no paradigma da orientação a objetos, é um tipo de classe que contém apenas as **assinaturas de métodos, propriedades, eventos e indexadores**.
- A implementação dos membros é feita por uma classe concreta que implementa a interface.

- Uma interface funciona como um **contrato** entre si e qualquer classe ou estrutura que a implementa. Isso significa que uma classe que implementa uma interface é **obrigada a implementar todos os seus membros**.
- Uma Interface tem apenas a declaração de **membro ou assinatura** e, implicitamente, todos os membros de uma interface são **públicos e abstratos**.

# Características

- Uma interface não fornece herança como uma classe ou classe abstrata, ela só declara membros que uma classe de implementação precisa implementar.
- Uma interface não pode ser instanciada mas pode ser referenciada pelo objeto da classe que a implementa. Além disso, a referência da interface funciona como objeto de referência e se comporta como o objeto.
- Uma interface não pode conter constantes, construtores, variáveis de instância, destrutores, membros estáticos ou interfaces aninhadas.



- Os membros de uma interface não pode ter qualquer modificador de acesso mesmo público.
- Implicitamente, cada membro de uma interface é público e abstrato. Além disso, você não tem permissão para especificar os membros de uma interface pública e abstratas ou virtuais.
- Uma interface pode ser herdada a partir de uma ou mais interfaces.
- Uma classe ou estrutura pode implementar mais de uma interface.
- Uma classe que implementa uma interface pode marcar qualquer método da interface como virtual e este método pode ser sobrescrito pelas classes derivadas.

# Quando usar Interfaces

- Necessidade de fornecer funcionalidade comum para as classes não relacionadas.
- Necessidade de agrupar objetos com base em comportamentos comuns.
- Necessidade de fornecer uma visão abstrata de um modelo que é imutável.
- Necessidade de criar componentes de baixo acoplamento, fácil manutenção e componentes conectáveis, visto que a implementação de uma interface é separada de si mesmo.

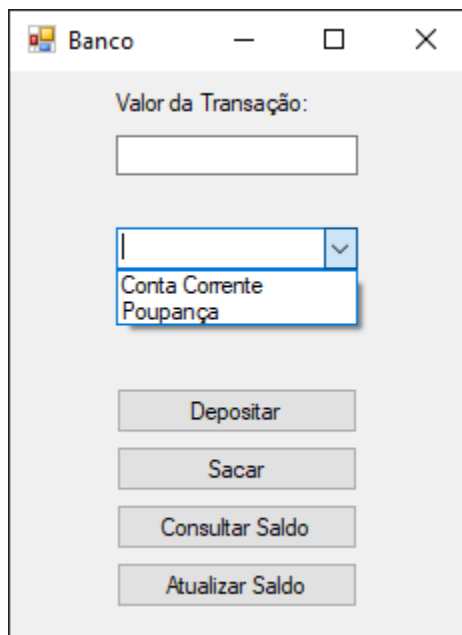
# Desvantagens

- O principal problema das interfaces é que quando você **adiciona novos membros a uma interface** você tem que **implementar esses membros em todas as classes que implementam esta interface**;
- Interfaces são **lentas** pois elas necessitam de um esforço extra para encontrar o método na classe atual;

# Refazendo o controle bancário com Interfaces

---

# Formulário



The image shows a screenshot of a web application window titled "Banco". The window contains a form for performing a transaction. The form has the following elements:

- A label "Valor da Transação:" followed by a text input field.
- A dropdown menu with a blue border and a downward arrow icon. The menu is open, showing two options: "Conta Corrente" and "Poupança".
- Four buttons stacked vertically: "Depositar", "Sacar", "Consultar Saldo", and "Atualizar Saldo".

# Código da Interface

```
2 references | 0 changes | 0 authors, 0 changes
interface IConta
{
    4 references | 0 changes | 0 authors, 0 changes
    void Debitar(double valor);

    4 references | 0 changes | 0 authors, 0 changes
    void Creditar(double valor);

    3 references | 0 changes | 0 authors, 0 changes
    void atualizarSaldos();
}
```

# Código da classe ContaCorrente

```
class ContaCorrente : IConta
{
    3 references | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change
    public double limiteEspecial { get; set; }
    11 references | 0 changes | 0 authors, 0 changes
    public double saldo { get; set; }

    0 references | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change
    public ContaCorrente()
    {
        this.limiteEspecial = 0;
        this.saldo = 0;
    }

    1 reference | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change
    public ContaCorrente(double saldo, double limiteEspecial)
    {
        this.limiteEspecial = limiteEspecial;
        this.saldo = saldo;
    }
}
```

```
public void Debitar(double valor)
{
    if (valor <= this.saldo + this.limiteEspecial)
    {
        saldo -= valor;
        MessageBox.Show("Saque efetuado");
    }
    else
    {
        MessageBox.Show("Saldo Insuficiente");
    }
}

2 references | 0 changes | 0 authors, 0 changes
public void Creditar(double valor)
{
    saldo += valor;
    MessageBox.Show("Depósito efetuado");
}

2 references | 0 changes | 0 authors, 0 changes
public void atualizarSalDOS()
{
    double saldoAnterior = this.saldo;
    if (this.saldo < 0)
    {
        saldo += saldo * 0.08;
    }
    MessageBox.Show("Saldo Anterior: " + saldoAnterior.ToString() +
        "\nSaldo Atual: " + this.saldo);
}
}
```

# Código da classe ContaPoupanca

```
class ContaPoupanca: IConta
{
    4 references | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change
    public double reajusteMensal { get; set; }
    11 references | 0 changes | 0 authors, 0 changes
    public double saldo { get; set; }

    0 references | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change
    public ContaPoupanca()
    {
        this.reajusteMensal = 0;
        this.saldo = 0;
    }

    1 reference | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change
    public ContaPoupanca(double saldo, double reajusteMensal)
    {
        this.saldo = saldo;
        this.reajusteMensal = reajusteMensal;
    }

    2 references | 0 changes | 0 authors, 0 changes
    public void Debitar(double valor)
    {
        if (valor <= this.saldo)
        {
            saldo -= valor;
            MessageBox.Show("Saque efetuado");
        }
        else
        {
            MessageBox.Show("Saldo Insuficiente");
        }
    }
}
```

```
public void Creditar(double valor)
{
    saldo += valor;
    MessageBox.Show("Depósito efetuado");
}
```

```
1 reference | 0 changes | 0 authors, 0 changes
public void atualizarSaldos()
{
}
```

```
1 reference | Alex Sander Resende de Deus, 22 hours ago | 1 author, 1 change
public void atualizarSaldos(double reajuste)
{
    this.reajusteMensal = reajuste;
    double saldoAnterior = this.saldo;
    this.saldo += this.saldo * (this.reajusteMensal / 100);
    MessageBox.Show("Saldo Anterior: " + saldoAnterior.ToString() +
        "\nSaldo Atual: " + this.saldo);
}
```



# Código dos botões

```
ContaCorrente cc1=new ContaCorrente(500,1000);  
ContaPoupanca cp1 = new ContaPoupanca(5000, 0.01);
```

1 reference | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change

```
private void btnDepositar_Click(object sender, EventArgs e)  
{  
    double valor = Convert.ToDouble(txtValor.Text);  
    switch (cmbTipoConta.SelectedIndex)  
    {  
        case 0:  
            cc1.Creditar(valor);  
            break;  
        case 1:  
            cp1.Creditar(valor);  
            break;  
    }  
}
```

```
private void btnSacar_Click(object sender, EventArgs e)  
{  
    double valor = Convert.ToDouble(txtValor.Text);  
    switch (cmbTipoConta.SelectedIndex)  
    {  
        case 0:  
            cc1.Debitar(valor);  
            break;  
        case 1:  
            if (valor <= cp1.saldo)  
            {  
                cp1.Debitar(valor);  
            }  
            else  
            {  
                MessageBox.Show("Saldo insuficiente");  
            }  
            break;  
    }  
}
```

1 reference | Alex Sander Resende de Deus, 4 days ago | 1 author, 1 change

```
private void btnConsultarSaldo_Click(object sender, EventArgs e)  
{  
    switch (cmbTipoConta.SelectedIndex)  
    {  
        case 0:  
            MessageBox.Show("Saldo em CC: R$" + cc1.saldo.ToString());  
            break;  
        case 1:  
            MessageBox.Show("Saldo em Poupança: R$" + cp1.saldo.ToString());  
            break;  
    }  
}
```

# Código dos botões

```
private void btnAtualizaSaldo_Click(object sender, EventArgs e)
{
    switch (cmbTipoConta.SelectedIndex)
    {
        case 0:
            cc1.atualizarSaldos();
            break;
        case 1:
            double reajuste = Convert.ToDouble(Interaction.InputBox("Digite a taxa de reajuste"));
            cp1.atualizarSaldos(reajuste);
            break;
    }
}
```



# Momento Hands On



Refazer o exercício **CONTROLE DE DESPESAS**  
transformando a **super classe** em uma  
**Interface.**

Vamos lá  
**Nosso último HAND ON!!!!**

Foi, de verdade, um **prazer** compartilhar estas  
semanas com vocês!

**Sucesso a todos!**

# OBRIGADO



profalex.deus@fiap.com.br



[linkedin.com/in/alexanderresende](https://www.linkedin.com/in/alexanderresende)

FIAP MBA<sup>+</sup>

Copyright © 2019 | Professor (a) Nome do Professor

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

FIAP