

Curso C# Completo

Programação Orientada a Objetos + Projetos

Capítulo: Herança e polimorfismo

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Herança

<http://educandoweb.com.br>

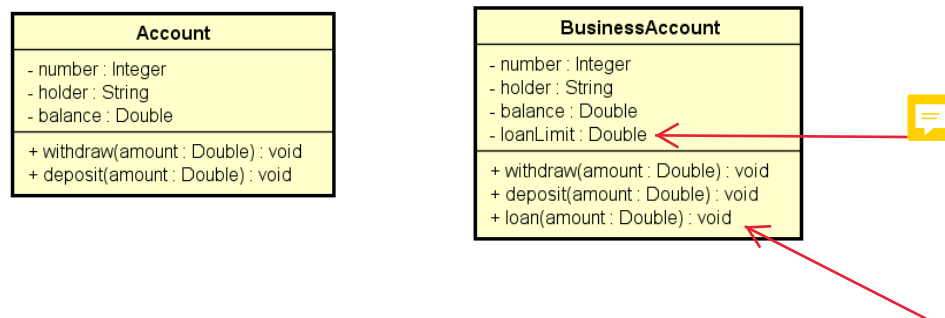
Prof. Dr. Nelio Alves

Herança

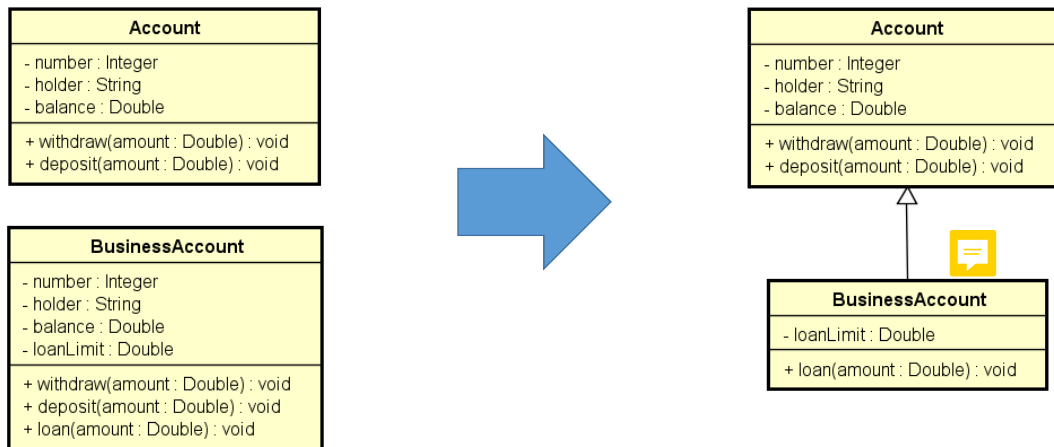
- É um tipo de associação que permite que uma classe herde dados e comportamentos de outra
- Definições importantes
- Vantagens
 - Reuso
 - Polimorfismo
- Sintaxe
 - : (estende) ←
 - base (referência para a superclasse)

Exemplo

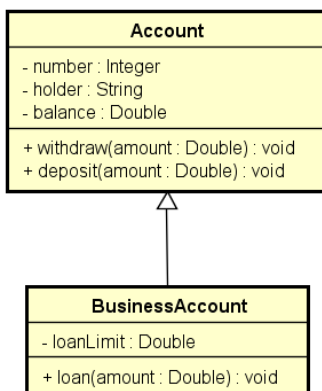
Suponha um negócio de banco que possui uma conta comum e uma conta para empresas, sendo que a conta para empresa possui todos membros da conta comum, mais um limite de empréstimo e uma operação de realizar empréstimo.



Herança permite o reuso de atributos e métodos (dados e comportamento)



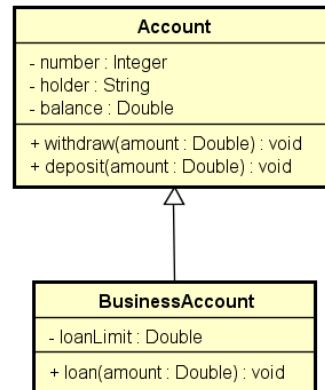
Definições importantes



- Relação "é-um"
- Generalização/especialização
- Superclasse (classe base) / subclasse (classe derivada)
- Herança / extensão
- Herança é uma associação entre classes (e não entre objetos)

Demo

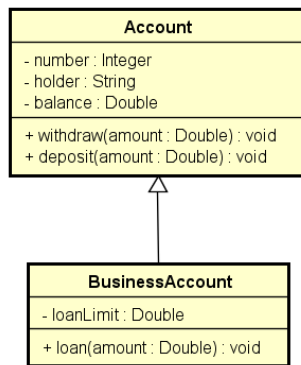
Vamos implementar as classes Account e BusinessAccount e fazer alguns testes.



Modificadores de acesso

	própria classe	subclasses no assembly	classes do assembly	subclasses fora do assembly	classes fora do assembly
public	x	x	x	x	x
protected internal	x	x	x	x	
internal	x	x	x		
protected	x	x		x	
private protected	x	x			
private	x				





Suponha que, para realizar um empréstimo, é descontada uma taxa no valor de 10.0

Isso resulta em erro:

```
public void Loan(double amount) {
    if (amount <= loanLimit) {
        balance += amount - 10.0;
    }
}
```

<https://github.com/acenelio/inheritance1-csharp>

Modificador de acesso protected

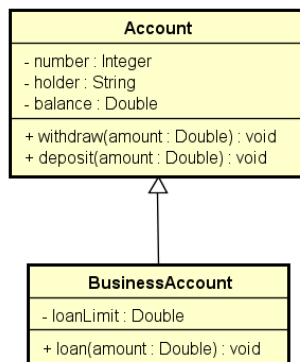
<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Membros

	própria classe	subclasses no assembly	classes do assembly	subclasses fora do assembly	classes fora do assembly
public	x	x	x	x	x
protected internal	x	x	x	x	
internal	x	x	x		
protected	x	x		x	
private protected	x	x			
private	x				

Problema exemplo



Se o saldo tiver acesso privativo para alteração, isso resulta em erro:

```
public void Loan(double amount)
{
    if (amount <= LoanLimit)
    {
        Balance += amount;
    }
}
```

<https://github.com/acenelio/inheritance1-csharp>

Upcasting e downcasting

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

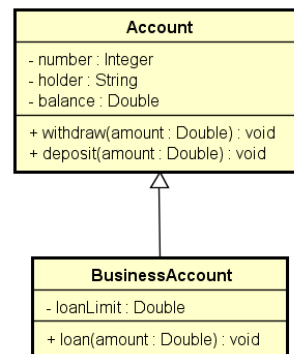
Checklist

- Upcasting

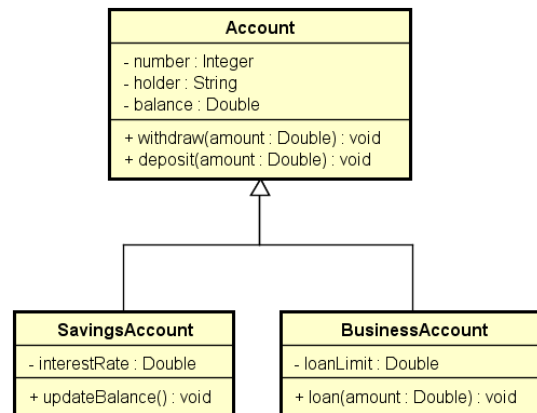
- Casting da subclasse para superclasse
- Uso comum: polimorfismo

- Downcasting

- Casting da superclasse para subclasse
- Palavra as
- Palavra is
- Uso comum: métodos que recebem parâmetros genéricos (ex: Equals)



Exemplo



<https://github.com/acenelio/inheritance2-csharp>

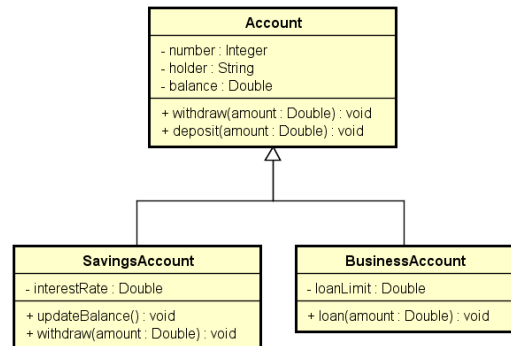
Sobreposição, palavras virtual, override e base

<http://educandoweb.com.br>

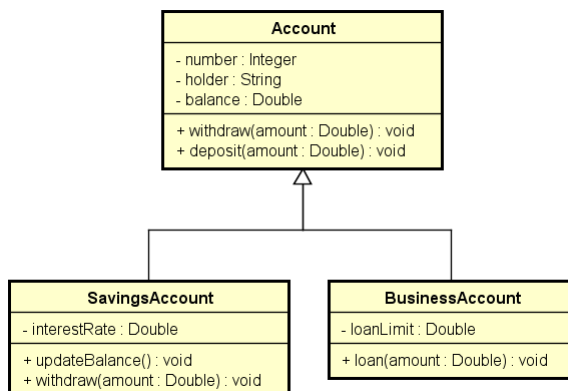
Prof. Dr. Nelio Alves

Sobreposição ou sobrescrita

- É a implementação de um método de uma superclasse na subclasse
- Para que um método comum (não abstrato) possa ser sobreposto, deve ser incluído nele o prefixo "**virtual**"
- Ao sobrescrever um método, devemos incluir nele o prefixo "**override**"



Exemplo



Suponha as seguintes regras para saque:

- Conta comum: é cobrada uma taxa no valor de 5.00.
- Conta poupança: não é cobrada taxa.

Como resolver isso?

Resposta: sobrescrevendo o método `withdraw` na subclasse **SavingsAccount**

Account:

```
public virtual void Withdraw(double amount) {  
    Balance -= amount + 5.0;  
}
```

SavingsAccount:

```
public override void Withdraw(double amount) {  
    Balance -= amount;  
}
```

Palavra base

É possível chamar a implementação da superclasse usando a palavra base.

Exemplo: suponha que a regra para saque para conta poupança seja realizar o saque normalmente da superclasse (Account), e depois descontar mais 2.0.

```
public override void Withdraw(double amount) {  
    base.Withdraw(amount);  
    Balance -= 2.0;  
}
```

Recordando: usando **base** em construtores

```
class BusinessAccount : Account
{
    public double LoanLimit { get; set; }

    public BusinessAccount()
    {
    }

    public BusinessAccount(int number, string holder, double balance, double loanLimit)
        : base(number, holder, balance)
    {
        LoanLimit = loanLimit;
    }

    (...)
}
```

Código fonte desta aula

<https://github.com/acenelio/inheritance3-csharp>

Classes e métodos selados

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Classes e métodos selados

- Palavra chave: sealed
- **Classe:** evita que a classe seja herdada
 - Nota: ainda é possível estender a funcionalidade de uma classe selada usando "extension methods"

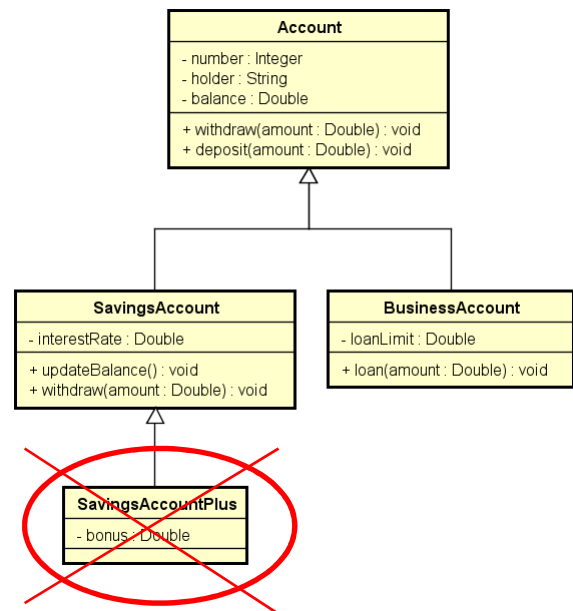
```
namespace Course {  
    sealed class SavingsAccount {
```

- **Método:** evita que um método sobreposto possa ser sobreposto novamente
 - Só pode ser aplicado a métodos sobrepostos

Exemplo - Classe selada

Suponha que você queira evitar que sejam criadas subclasses de SavingsAccount

```
namespace Course {  
    sealed class SavingsAccount {  
  
        (...)  
    }  
}
```



Exemplo - método selado

Suponha que você não queira que o método Withdraw de SavingsAccount seja sobreposto novamente

```
public sealed override void Withdraw(double amount)  
{  
    base.Withdraw(amount);  
    Balance -= 2.0;  
}
```

Pra quê?

- Segurança: dependendo das regras do negócio, às vezes é desejável garantir que uma classe não seja herdada, ou que um método não seja sobreposto.
 - Geralmente convém selar métodos sobrepostos, pois sobreposições múltiplas podem ser uma porta de entrada para inconsistências
- Performance: atributos de tipo de uma classe selada são analisados de forma mais rápida em tempo de execução.
 - Exemplo clássico: string

Introdução ao polimorfismo

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Pilares da OOP

- Encapsulamento
- Herança
- Polimorfismo

Polimorfismo

Em Programação Orientada a Objetos, polimorfismo é recurso que permite que variáveis de um mesmo tipo mais genérico possam apontar para objetos de tipos específicos diferentes, tendo assim comportamentos diferentes conforme cada tipo específico.

```
Account acc1 = new Account(1001, "Alex", 500.0);  
Account acc2 = new SavingsAccount(1002, "Anna", 500.0, 0.01);  
  
acc1.Withdraw(10.0);  
acc2.Withdraw(10.0);  
  
Console.WriteLine(acc1.Balance);  
Console.WriteLine(acc2.Balance);
```

```
Account acc1 = new Account(1001, "Alex", 500.0);
Account acc2 = new SavingsAccount(1002, "Anna", 500.0, 0.01);

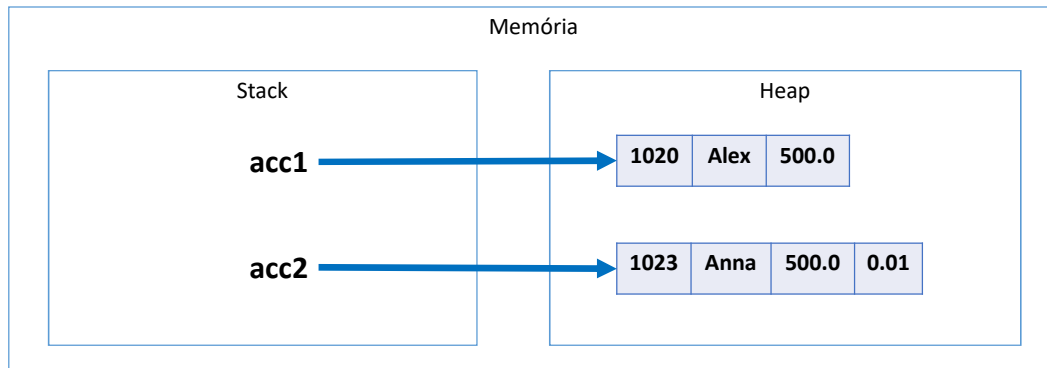
acc1.Withdraw(10.0);
acc2.Withdraw(10.0);
```

Account:

```
public virtual void Withdraw(double amount) {
    Balance -= amount + 5.0;
}
```

SavingsAccount:

```
public override void Withdraw(double amount) {
    base.Withdraw(amount);
    Balance -= 2.0;
}
```



Importante entender

- A associação do tipo específico com o tipo genérico é feita **em tempo de execução** (upcasting).
- O compilador não sabe para qual tipo específico a chamada do método Withdraw está sendo feita (ele só sabe que são duas variáveis tipo Account):

```
Account acc1 = new Account(1001, "Alex", 500.0);
Account acc2 = new SavingsAccount(1002, "Anna", 500.0, 0.01);

acc1.Withdraw(10.0);
acc2.Withdraw(10.0);
```


Exercício resolvido

<http://educandoweb.com.br>

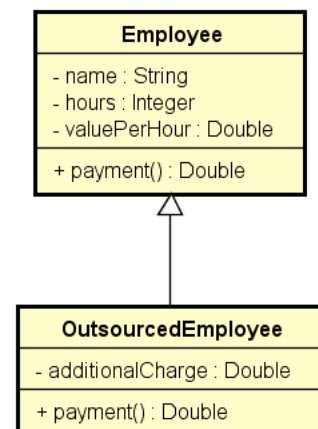
Prof. Dr. Nelio Alves

Uma empresa possui funcionários próprios e terceirizados. Para cada funcionário, deseja-se registrar nome, horas trabalhadas e valor por hora. Funcionários terceirizados possuem ainda uma despesa adicional.

O pagamento dos funcionários corresponde ao valor da hora multiplicado pelas horas trabalhadas, sendo que os funcionários terceirizados ainda recebem um bônus correspondente a 110% de sua despesa adicional.

Fazer um programa para ler os dados de N funcionários (N fornecido pelo usuário) e armazená-los em uma lista. Depois de ler todos os dados, mostrar nome e pagamento de cada funcionário na mesma ordem em que foram digitados.

Construa o programa conforme projeto ao lado. Veja exemplo na próxima página.



Enter the number of employees: **3**

Employee #1 data:

Outsourced (y/n)? **n**

Name: **Alex**

Hours: **50**

Value per hour: **20.00**

Employee #2 data:

Outsourced (y/n)? **y**

Name: **Bob**

Hours: **100**

Value per hour: **15.00**

Additional charge: **200.00**

Employee #3 data:

Outsourced (y/n)? **n**

Name: **Maria**

Hours: **60**

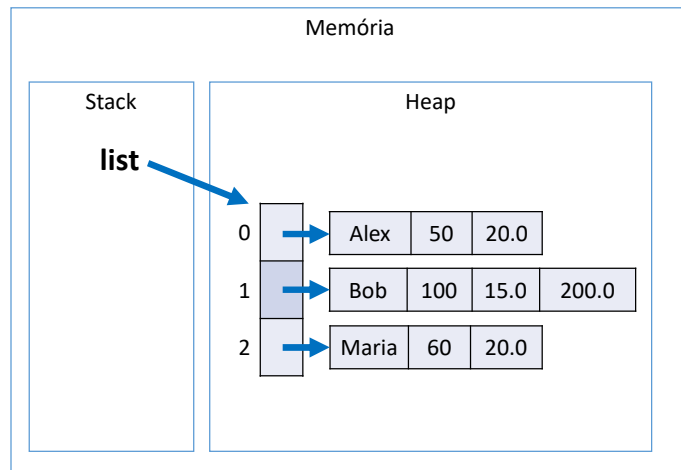
Value per hour: **20.00**

PAYMENTS:

Alex - \$ 1000.00

Bob - \$ 1720.00

Maria - \$ 1200.00



<https://github.com/acenelio/inheritance4-csharp>



Exercício de fixação

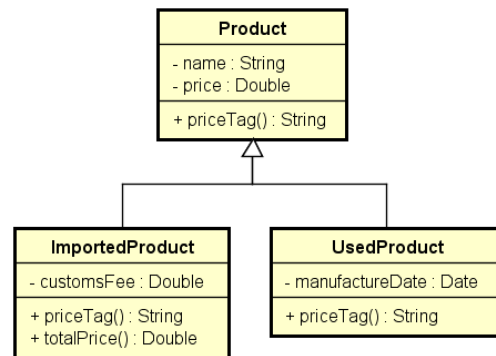
<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler os dados de N produtos (N fornecido pelo usuário). Ao final, mostrar a etiqueta de preço de cada produto na mesma ordem em que foram digitados.

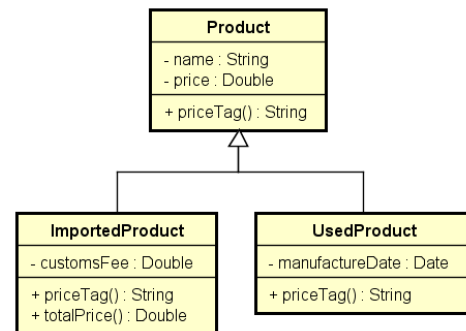
Todo produto possui nome e preço. Produtos importados possuem uma taxa de alfândega, e produtos usados possuem data de fabricação. Estes dados específicos devem ser acrescentados na etiqueta de preço conforme exemplo (próxima página). Para produtos importados, a taxa e alfândega deve ser acrescentada ao preço final do produto.

Favor implementar o programa conforme projeto ao lado.



```
Enter the number of products: 3
Product #1 data:
Common, used or imported (c/u/i)? i
Name: Tablet
Price: 260.00
Customs fee: 20.00
Product #2 data:
Common, used or imported (c/u/i)? c
Name: Notebook
Price: 1100.00
Product #3 data:
Common, used or imported (c/u/i)? u
Name: Iphone
Price: 400.00
Manufacture date (DD/MM/YYYY): 15/03/2017
```

```
PRICE TAGS:
Tablet $ 280.00 (Customs fee: $ 20.00)
Notebook $ 1100.00
Iphone (used) $ 400.00 (Manufacture date: 15/03/2017)
```



<https://github.com/acenelio/inheritance5-csharp>

Classes abstratas

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Classes abstratas

- São classes que não podem ser instanciadas
- É uma forma de garantir herança total: somente subclasses não abstratas podem ser instanciadas, mas nunca a superclasse abstrata

Exemplo

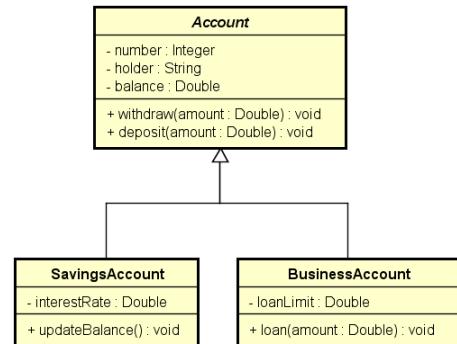
Suponha que em um negócio relacionado a banco, apenas contas poupança e contas para empresas são permitidas. Não existe conta comum.

Para garantir que contas comuns não possam ser instanciadas, basta acrescentarmos a palavra "abstract" na declaração da classe.

```
namespace Course {  
    abstract class Account {  
        (...)  
    }  
}
```

Notação UML: *itálico*

Vamos partir da implementação em: <https://github.com/acenelio/inheritance3-csharp>



Questionamento

- Se a classe **Account** não pode ser instanciada, por que simplesmente não criar somente **SavingsAccount** e **BusinessAccount**?

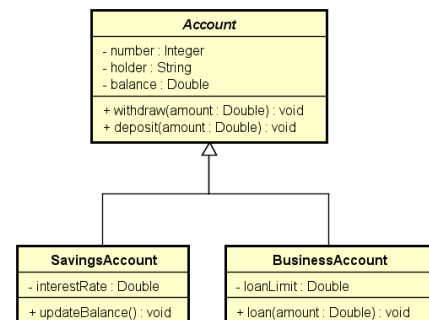
- Resposta:

- **Reuso**
- **Polimorfismo**: a superclasse classe genérica nos permite tratar de forma fácil e uniforme todos os tipos de conta, inclusive com polimorfismo se for o caso (como fizemos nos últimos exercícios). Por exemplo, você pode colocar todos tipos de contas em uma mesma coleção.

- Demo: suponha que você queira:

- Totalizar o saldo de todas as contas.
- Sacar 10.00 de todas as contas.

<https://github.com/acenelio/inheritance6-csharp>



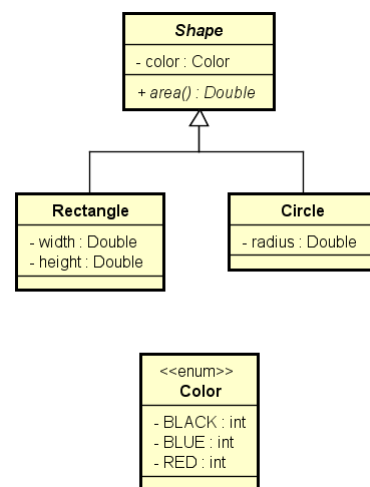
Métodos abstratos

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Métodos abstratos

- São métodos que não possuem implementação.
- Métodos precisam ser abstratos quando a classe é genérica demais para conter sua implementação.
- Se uma classe possuir pelo menos um método abstrato, então esta classe também é abstrata.
- Notação UML: itálico
- Exercício resolvido



Exercício resolvido (métodos abstratos)

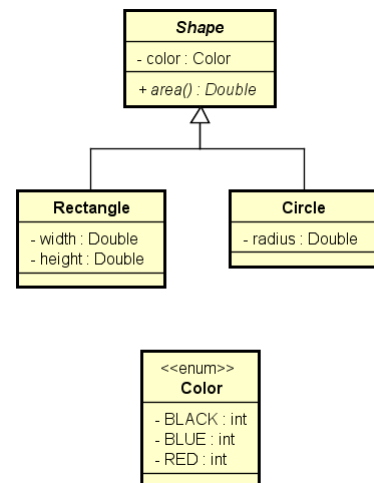
<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler os dados de N figuras (N fornecido pelo usuário), e depois mostrar as áreas destas figuras na mesma ordem em que foram digitadas.

```
Enter the number of shapes: 2
Shape #1 data:
Rectangle or Circle (r/c)? r
Color (Black/Blue/Red): Black
Width: 4.0
Height: 5.0
Shape #2 data:
Rectangle or Circle (r/c)? c
Color (Black/Blue/Red): Red
Radius: 3.0

SHAPE AREAS:
20.00
28.27
```



<https://github.com/acenelio/inheritance7-csharp>

Exercício de fixação

<http://educandoweb.com.br>

Prof. Dr. Nelio Alves

Fazer um programa para ler os dados de **N contribuintes** (N fornecido pelo usuário), os quais podem ser **pessoa física ou pessoa jurídica**, e depois mostrar o **valor do imposto pago por cada um**, bem como o **total de imposto arrecadado**.

Os dados de **pessoa física** são: **nome, renda anual e gastos com saúde**. Os dados de **pessoa jurídica** são **nome, renda anual e número de funcionários**. As regras para cálculo de imposto são as seguintes:

Pessoa física: pessoas cuja **renda foi abaixo de 20000.00** pagam **15% de imposto**. Pessoas com renda de **20000.00 em diante** pagam **25% de imposto**. Se a pessoa teve **gastos com saúde**, **50% destes gastos são abatidos no imposto**.

Exemplo: uma pessoa cuja renda foi 50000.00 e teve 2000.00 em gastos com saúde, o imposto fica: $(50000 * 25\%) - (2000 * 50\%) = 11500.00$

Pessoa jurídica: **pessoas jurídicas pagam 16% de imposto**. Porém, se a **empresa possuir mais de 10 funcionários**, ela paga **14% de imposto**.

Exemplo: uma empresa cuja renda foi 400000.00 e possui 25 funcionários, o imposto fica: $400000 * 14\% = 56000.00$

Enter the number of tax payers: **3**

Tax payer #1 data:

Individual or company (i/c)? **i**

Name: **Alex**

Annual income: **50000.00**

Health expenditures: **2000.00**

Tax payer #2 data:

Individual or company (i/c)? **c**

Name: **SoftTech**

Annual income: **400000.00**

Number of employees: **25**

Tax payer #3 data:

Individual or company (i/c)? **i**

Name: **Bob**

Annual income: **120000.00**

Health expenditures: **1000.00**

TAXES PAID:

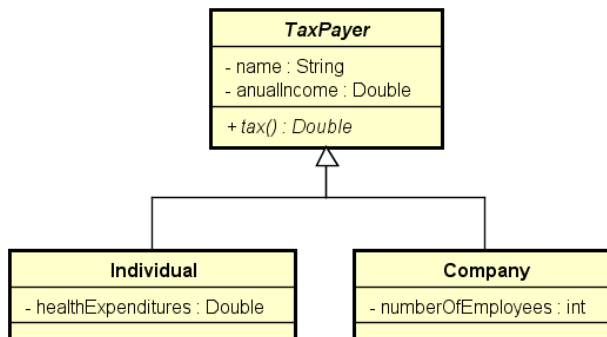
Alex: \$ 11500.00

SoftTech: \$ 56000.00

Bob: \$ 29500.00

TOTAL TAXES: \$ 97000.00

Correção



<https://github.com/acenelio/inheritance8-csharp>