**CONTA**

Tipos de Conta: Conta Corrente, Poupança e Salário

Saldo inicial: R$ 100,0- OK

Total de Contas Criadas: Informar por tipo de conta o total de contas criadas.  
Em Andamento. Dificuldade em acessar a classe principal e colocar um contador dentro do construtor.

**Campos:**

* + Titular: obrigatório- OK
  + Agencia: obrigatório- OK
  + Número da conta: obrigatórios- OK

**Funções e Restrições**

* **Método Depositar**
  + Não pode depositar valores negativos- OK
* **Método Sacar**
  + Taxa de R$ 0,10 por saque no caso de Conta Corrente - OK
  + Não pode sacar mais que o valor do saldo- OK
  + Não pode sacar valores negativos- OK
  + **Método Transferir**
    - Não pode transferir mais que o valor do saldo- OK
    - Não pode transferir valores negativos- OK
    - Criar parâmetro para permitir ou não transferência entre Conta Corrente e Poupança - PEN

Solução:

* Criada uma classe abstrata Conta com os métodos abstratos: Sacar, Depositar e Transferir
* Criada classe contaCorrente, contaPoupança e contaSalario herdada de Conta
* O método Sacar foi feito com retorno Booleano, para poder ser usado como validação no método Transferir
* Na classe contaCorrente, foi criado um método **sacarRecursoSemTaxa** para ser usado no método **Transferir** sem a cobrança automática de Taxa
* Precisa ter uma propriedade que fará a soma de todas as contas criadas.
  + Este número só pode ser obtido. Não pode ser setado.
  + Dificuldas encontrada: Não estou conseguindo acessar a classeAbstrata para fazer uma contagem geral. So é possível pelo objeto instanciado.
* Uma propriedade para calcular a taxa de Operação.
  + A regra será: Quanto mais contas criadas, menor a taxa de operação.
  + Este número só pode ser obtido. Não pode ser setado.
* Criar as propriedades Agencia e Conta - OK
* Agencia e Conta precisa ser read Only
  + Resolvido com variável do tipo val
* val cc = contaCorrente("Conta Corrente",1234,4444)
* Criar propriedade Titular do Tipo Cliente
* Construtor da Classe Conta Corrente
* Obrigatório informar agencia e conta.
* Validação do número de agencia e conta não poder ser menor que zero. (Usar Argument Excpetion)
* Chamar as propriedades de Criação de conta e Cálculo de Taxa.

METODO SALDO

* O saldo inicial padrão de todas as contas é R$100,00
* Este saldo nunca pode ficar negativo.

**METODO SACAR**

* Argumento do Método: Valor a ser sacado.
* Validar se o valor a ser sacado não é negativo.
* Validar se há saldo antes de realizar o saque.
* Para as exceções, criar uma Classe específica de Saldo Insuficiente, herdada da Classe Expection
* A exceção deve lançar a mensagem com valor de saque pretendido e valor de saldo atual.

**Metodo Transferir**

* Metodo transferir usa como argumentos a conta destino e o valor a ser transferido
* Validar se o valor à ser transferido não é menor que 01. Lançar exceção com nome do parâmetro.
* A validação do valor a ser transferido vai ser toda feita pelo método Sacar, que já prever consulta ao saldo
* Com tudo ok, chamar o Método Depositar, que fará o depósito.

**Método Depositar**

* Método Depositar com argumento do valor à ser depositado.
* Adiciona o valor à ser Depositado à conta.

**CLASSE EXCEÇÃO - SALDO INSUFICIENTE**

* Herdar a classe Exception  
  Criar propriedades de Saldo e Valor do Saque para compor a mensagem de exceção.   
  Construtor simples da Classe, sem argumentos  
  Construtor da Classe com utilização da mensagem no construtor  
  Construtor da Classe que utilizará as propriedades Saldo e ValorSaque para compor a mensagem de exceção.

**CLASSE CLIENTE**

* Propriedade Nome .   
  Propriedade CPF  
  Propriedade Profissao  
  Contator de clientes criado
* Exigir no Construtor a informação de nome e CPF e somar o total de clientes criados

**CLASSE FUNCIONARIO**

* A classe funcionário será abstrata [...]
* Propriedade Nome  
  Propriedade Salario com SET protegido [...]
* Propriedade CPF com SET privado.   
  Contator de Funcionarios criados, com SET privado  
  Construtor da classe indica obrigatoriedade dos campos Nome e CPF, e somando o funcionário criado  
  Método para aumento de salario abstrato [...]  
  Metodo para receber Bonificação como abstrato [...]

**CLASSE AUXILIAR : FUNCIONARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Funcionário | Salário Base | Aumento Salarial | Bonificação |
| Auxiliar | R$2.000 | 10% | 20% |

* Construtor da classe obedece a classe principal / base (Funcionario) e obriga o preenchimento de Nome e CPF
* Método AumentaSalario sobrescreve o método base com aumento de 10%. Propriedade Salario pertence à classe base.
* Método GetBonificação sobrescreve o método base com um retorno de 20% sobre o salário. Propriedade Salario pertence à classe base.

**CLASSE DESENVOLVEDOR: FUNCIONARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Funcionário | Salário Base | Aumento Salarial | Bonificação |
| Desenvolvedor | R$3.000 | 15% | 10% |

* Construtor da classe obedece a classe principal / base (Funcionario) e obriga o preenchimento de Nome e CPF  
  Método AumentaSalario sobrescreve o método base com aumento de 15%. Propriedade Salario pertence à classe base.  
  Método GetBonificação sobrescreve o método base com um retorno de 10% sobre o salário. Propriedade Salario pertence à classe base.

**CLASSE DESIGNER : FUNCIONARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Funcionário | Salário Base | Aumento Salarial | Bonificação |
| Designer | R$3.000 | 11% | 17% |

Construtor da classe obedece a classe principal / base (Funcionario) e obriga o preenchimento de Nome e CPF  
Método AumentaSalario sobrescreve o método base com aumento de 11%. Propriedade Salario pertence à classe base.  
Método GetBonificação sobrescreve o método base com um retorno de 17% sobre o salário. Propriedade Salario pertence à classe base.

**CLASSE DIRETOR : FUNCIONARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Funcionário | Salário Base | Aumento Salarial | Bonificação |
| Diretor | R$5.000 | 15% | 50% |

Construtor da classe obedece a classe principal / base (Funcionario) e obriga o preenchimento de Nome e CPF  
Método AumentaSalario sobrescreve o método base com aumento de 15%. Propriedade Salario pertence à classe base.

Método GetBonificação sobrescreve o método base com um retorno de 50% sobre o salário. Propriedade Salario pertence à classe base.

**CLASSE GERENTE DE CONTA : FUNCIONARIO**

|  |  |  |  |
| --- | --- | --- | --- |
| Funcionário | Salário Base | Aumento Salarial | Bonificação |
| Gerente de Conta | R$4.000 | 5% | 25% |

* Construtor da classe obedece a classe principal / base (Funcionario) e obriga o preenchimento de Nome e CPF
* Método AumentaSalario sobrescreve o método base com aumento de 5%. Propriedade Salario pertence à classe base.  
  Método GetBonificação sobrescreve o método base com um retorno de 25% sobre o salário. Propriedade Salario pertence à classe base.
* CLASSE PARCEIRO COMERCIAL : FUNCIONARIO
* CLASSE GERENCIADOR DE BONIFICAÇÃO
* SISTEMA INTERNO –
* CLASSE FUNCIONARIO AUTENTICAVEL
* CLASSE IAUTENTICAVEL
* CLASSE SISINTERNO
* CLASSE PROGRAM

Tecnologias

* Reuse código com Herança e Composição OK
* Use os modificadores virtual, abstract e override OK
* Use o modificador de visibilidade protected OK

|  |  |  |  |
| --- | --- | --- | --- |
| Funcionário | Salário Base | Aumento Salarial | Bonificação |
| Auxiliar | R$2.000 OK | 10% OK | 20% OK |
| Desenvolvedor | R$3.000 OK | 15% OK | 10% OK |
| Designer | R$3.000 OK | 11% OK | 17% OK |
| Diretor | R$5.000 OK | 15% OK | 50% OK |
| Gerente de Conta | R$4.000 OK | 5% OK | 25% OK |

Implementar Interface e métodos abstratos

Desenvolver um Sistema interno, autenticável apenas pelo Gerente de Contas, Diretor e Parceiro Comercial

Há também um tipo que deve autenticar: Parceiro Comercial. Este não é um funcionário   
[Conceito de Herança múltipla Interface]

Desafio Interface:

Utilizamos as interfaces no contexto de funcionários, porém, podemos também estender o exemplos para o contexto de contas. Por exemplo, na Bytebank pode existir um novo tipo de conta chamada de **conta salário**.

Esse tipo de conta costuma ter as mesmas informações e comportamentos que uma conta comum, porém com mais restrições. Por exemplo, não é possível usar o serviço de transferência.

Considerando esse detalhe, aceite o desafio e crie a **conta salário** que tem tudo que uma conta tem, com exceção do comportamento de transferir. Nesta implementação, mantenha a mesma técnica de polimorfismo na transferência para contas que precisam desse comportamento (corrente e poupança).

Para testar o código, tente realizar a transferência com a conta salário e confira se apresenta um problema de compilação.