FeMASS
FACULDADE PROF. MIGUEL ÂNGELO DA SILVA SANTOS

Projeto de Sistemas I

Faculdade Prof. Miguel Ângelo da Silva Santos

Material 7 – Orientação a Objetos em Java (Encapsulamento)

Professor: Isac Mendes Lacerda, M.Sc., PMP, CSM
e-mail: isac.curso@gmail.com

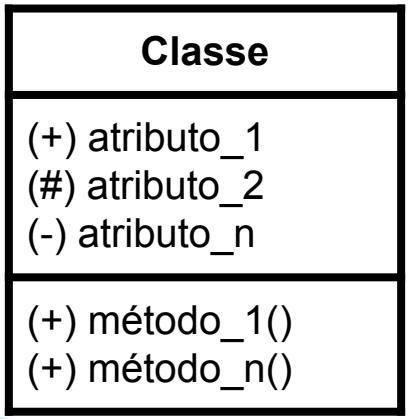
Tópicos

- Encapsulamento

Encapsulamento

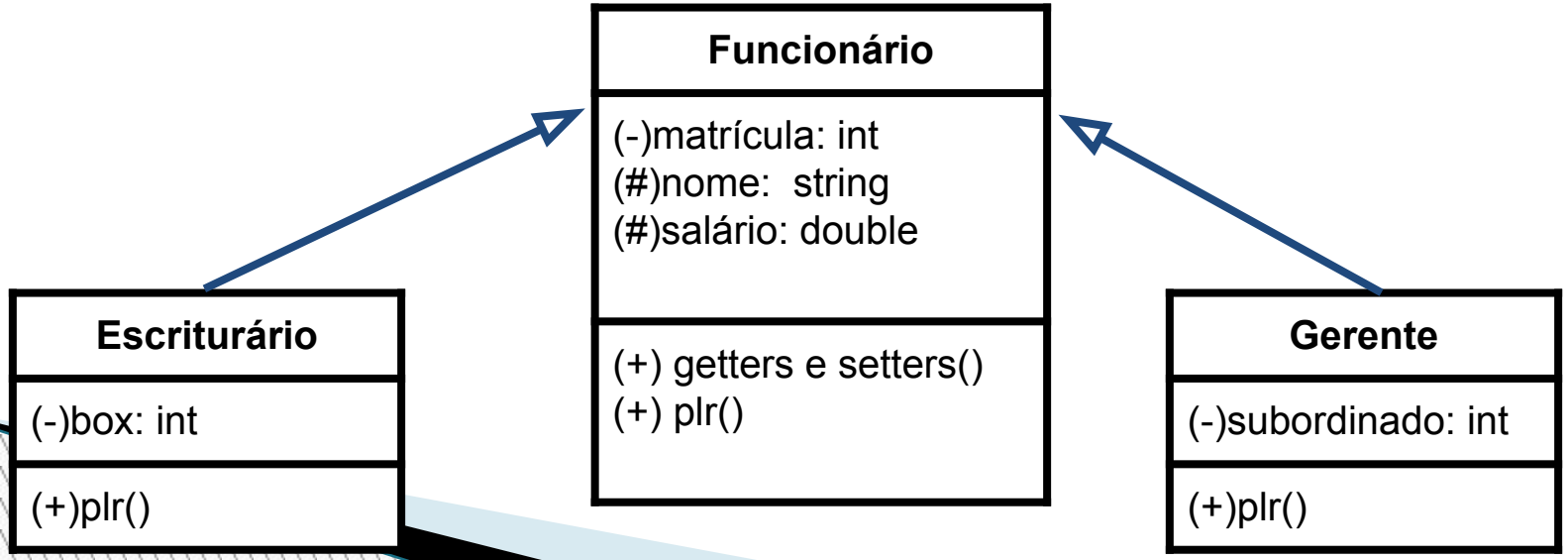
- Sei que você também não esqueceu que encapsular é uma forma de alterar a zona de visibilidade de métodos e atributos.

- E a finalidade disso é proteger você mesmo, como desenvolvedor, para manter a aplicação.



Encapsulamento

- Conceito geral:
  - (+) ***Public***: visível e manipulável dentro e fora da classe de definição.
  - (#) ***Protected***: visível e manipulável dentro da classe de definição e eventuais subclasses.
  - (-) ***Private***: visível e manipulável somente dentro da classe de definição.

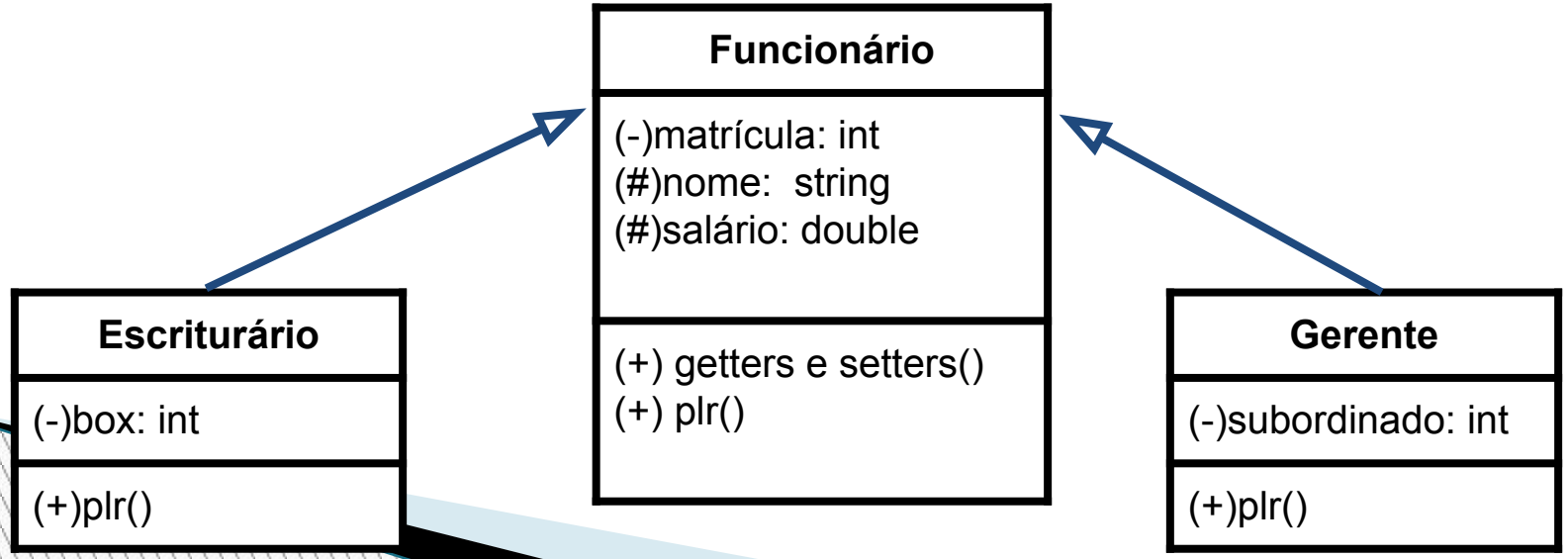


Encapsulamento

- **Mas em Java**, os níveis de visibilidade são:
  - (+) ***Public***: todos podem acessar aquilo que for definido como *public*. Classes, atributos, construtores métodos podem ser *public*.
  - (#) ***Protected***: pode ser acessado em todas as classes do mesmo pacote e em todas as classes que o estendam, mesmo que nesse último caso, essas não estejam no mesmo pacote. Somente atributos, construtores e métodos podem ser *protected*.
  - (-) ***Private***: visível e manipulável somente dentro da classe de definição.
  - ***Default***: (sem nenhum modificador) - Se nenhum modificador for utilizado, todas as classes do mesmo pacote têm acesso ao atributo, construtor, método ou classe.

Encapsulamento

- Os **elementos públicos** de uma classe são chamados de **interface da classe**. Expressam pontes de acesso.
- Tudo que estiver encapsulado é **alterado (mantido) apenas em um lugar**.
- Com **encapsulamento** todas **as referências**, através da interface, **desconhecem a alteração** na fonte.



Encapsulamento

```
    public class Empresa {
13      int cnpj;
14      String razao_social ="Padrão";
15      protected String nome_fantasia = "Padrão";
16      private String reponsavel = "Padrão";
17
18      void m1(){
19          System.out.println("Sou o método 1");
20      }
21
22      protected void m2(){
23          System.out.println("Sou o método 2");
24      }
25
26      private void m3(){
27          System.out.println("Sou o métodos 3");
28      }
29
30  }
```

```
12  public class EmpresaFranqueada extends Empresa{
13      String franqueador;
14      double percentual_franqueador;
15  }
```

Demonstração complementar Netbeans...

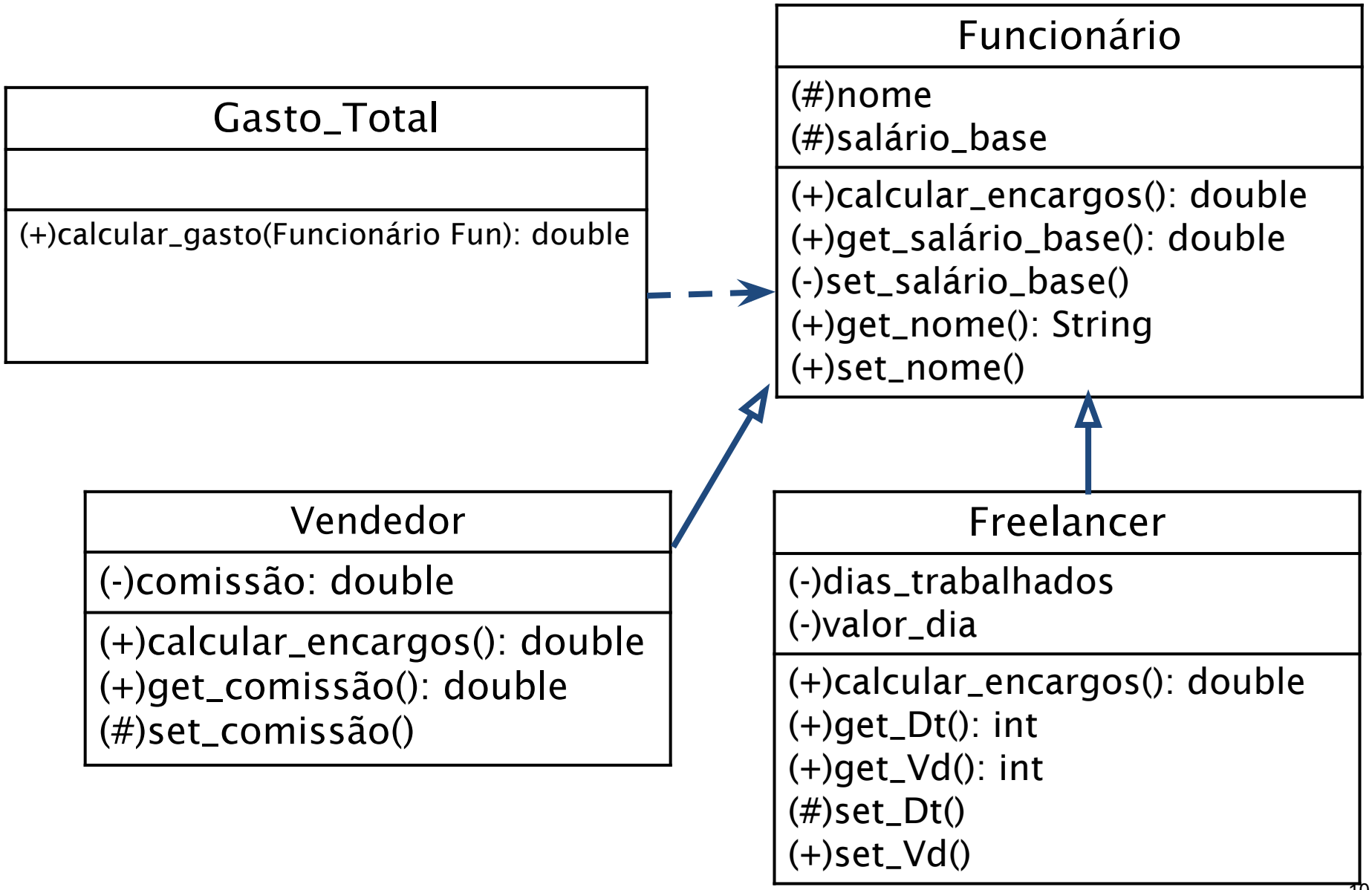
Exercício 1

Desta vez, **considere a indicação de encapsulamento,** implemente o modelo do próximo slide e crie um objeto de cada classe, testando todos os seus respectivos métodos e os efeitos do encapsulamento. **Propositalmente existem equívocos relacionados ao encapsulamento no modelo, identifique quais são e os corrija.**

Observações:

- Os métodos *calcular_encargos()*, contidos nas classes de funcionários devem ter diferenças de lógica (seja criativo), para justificar a sobrescrição.
- Na classe *Gasto_Total*, o método *calcular_gasto()* deve receber como argumento qualquer tipo de funcionário e contabilizar todo o valor gasto com esse funcionário (salário e encargos). A relação entre a classe *Gasto_Total* e *Funcionário* é do tipo dependência.

Exercício 1


Gasto_Total
(+)calcular_gasto(Funcionário Fun): double
Funcionário
(#)nome
(#)salário_base
(+)calcular_encargos(): double
(+)get_salário_base(): double
(-)set_salário_base()
(+)get_nome(): String
(+)set_nome()
Vendedor
(-)comissão: double
(+)calcular_encargos(): double
(+)get_comissão(): double
(#)set_comissão()
Freelancer
(-)dias_trabalhados
(-)valor_dia
(+)calcular_encargos(): double
(+)get_Dt(): int
(+)get_Vd(): int
(#)set_Dt()
(+)set_Vd()

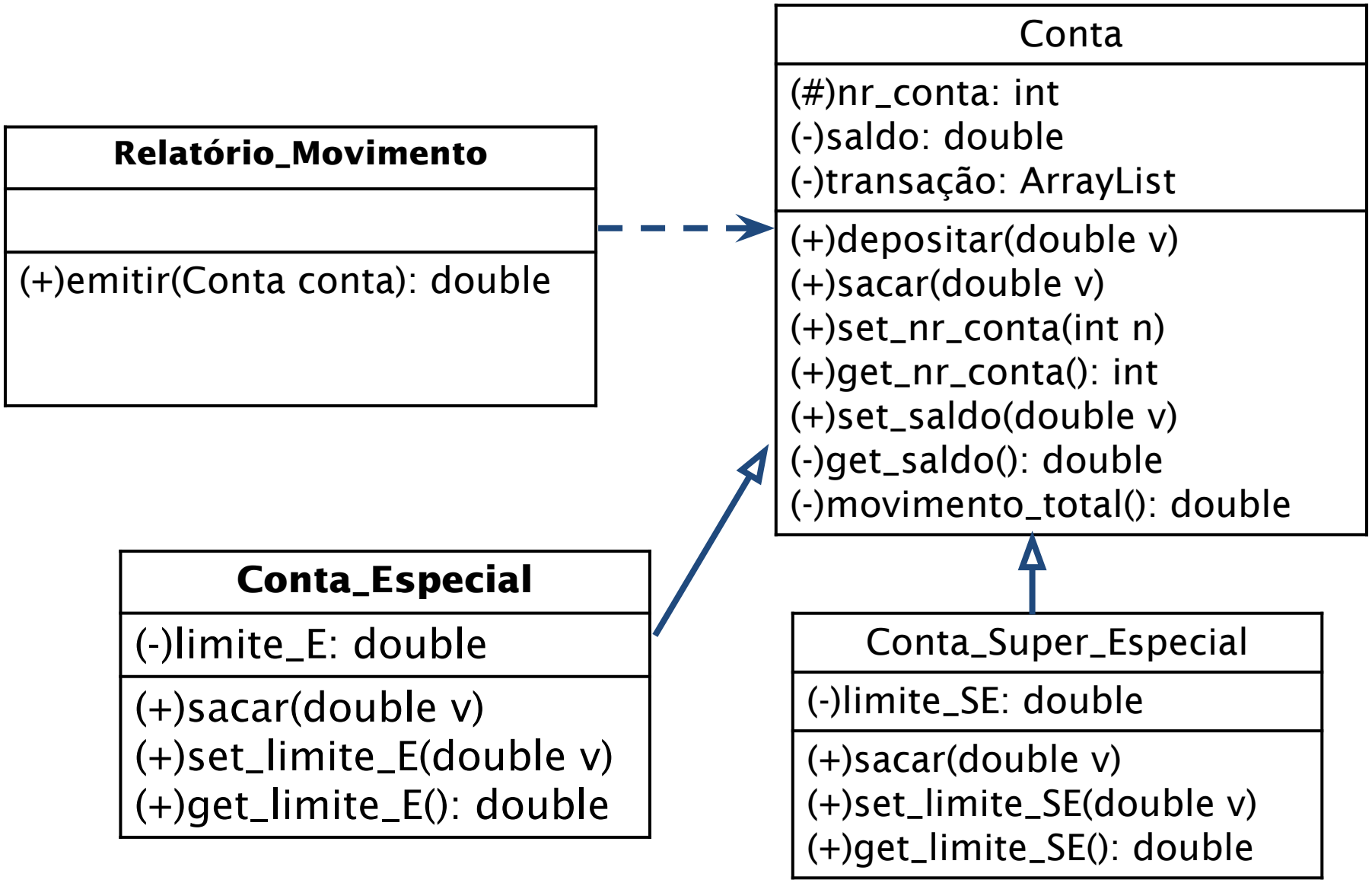
Exercício 2

Desta vez, **considere a indicação de encapsulamento**, implemente o modelo do próximo slide e crie um objeto de cada classe, testando todos os seus respectivos métodos e os efeitos do encapsulamento. **Propositalmente existem equívocos relacionados ao encapsulamento no modelo, identifique quais são e os corrija.**

Observações:

- Na classe *Conta_Genérica*, o atributo *Transação* deve armazenar cada valor movimentado (de saque ou depósito, de modo a manter o histórico).
- Na classe *Relatório_Movimento*, o método *emitir()* deve receber como argumento qualquer tipo de conta e contabilizar todo o valor da sua movimentação. A relação entre a classe *Gasto_Total* e *Funcionário* é do tipo dependência.

Exercício 2


Conta
(#)nr_conta: int
(-)saldo: double
(-)transação: ArrayList
(+)depositar(double v)
(+)sacar(double v)
(+)set_nr_conta(int n)
(+)get_nr_conta(): int
(+)set_saldo(double v)
(-)get_saldo(): double
(-)movimento_total(): double
Relatório_Movimento
(+)emitir(Conta conta): double
Conta_Especial
(-)limite_E: double
(+)sacar(double v)
(+)set_limite_E(double v)
(+)get_limite_E(): double
Conta_Super_Especial
(-)limite_SE: double
(+)sacar(double v)
(+)set_limite_SE(double v)
(+)get_limite_SE(): double