Programação para Web I
Aula 4

Profa. Silvia Bertagnolli

Modificador final

Classe final

Classe atingiu o nível máximo de especialização e não poderá mais ser especializada - nenhuma outra classe jamais poderá estender esta classe

Quando usar? Garantir que nenhum método da classe será sobreposto

Exemplo: java.lang.String

Classe final

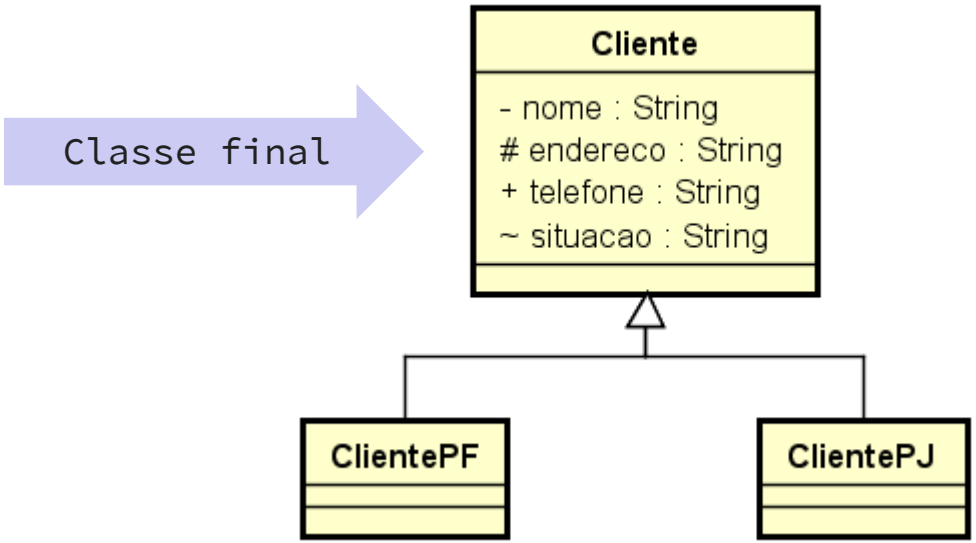
Vantagens:

- Permite **proteger** um código
- Aumenta o desempenho do código

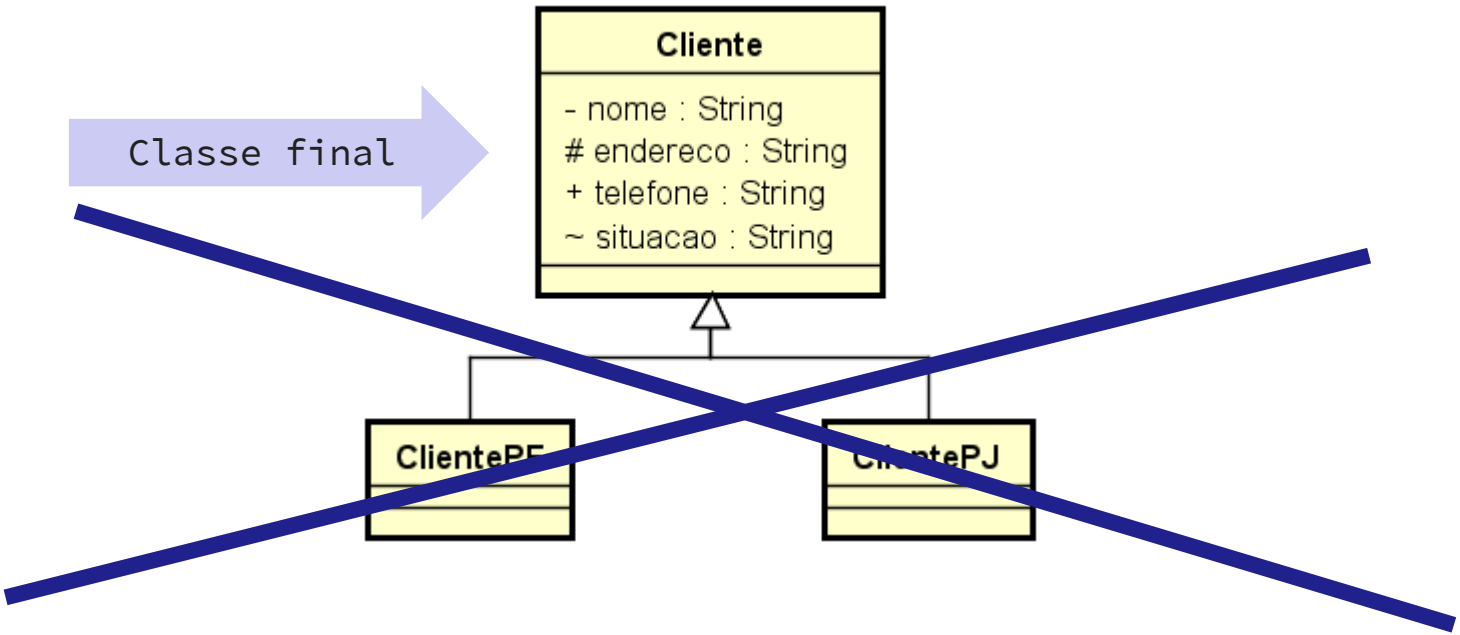
Desvantagem - **reduz** as possibilidades de **herança**

Classe final

Nenhuma outra classe jamais poderá estender esta classe



Classe final



Classe final: sintaxe

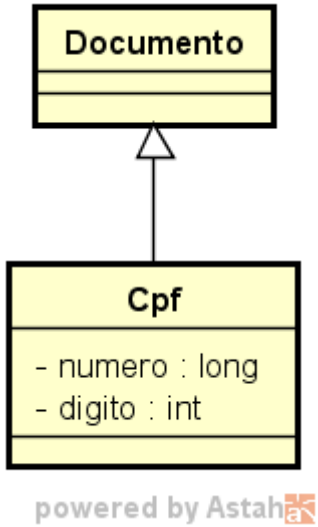
```
<modificador> final class <nome_classe>{

}
```

Classe final

```
public final class Documento{
     public boolean valida() {
               // corpo método
               return true;
      }
}

public class Cpf extends Documento{
....
}
```



?

Classe final

```
public class TesteFinal1{

        public static void main(String args[]) {

            Documento d = new Documento();

            if(d.valida())

                System.out.println("Documento válido");

        }

}
```

Método final

Método que **não** pode ser **sobrescrito** nas subclasses

Isso oferece **segurança** e **proteção**

Método declarado como final terá o seu protótipo sempre como foi definido e quando chamado por outros objetos seu código será executado

O que é sobrescrita e o que é sobrecarga?

Método final

O desempenho de execução de um método final é maior, pois as chamadas são substituídas pelo código contido na definição do método

“[...] se um método possuir uma especificação bem definida e não for sofrer especializações/redefinições pelas classes herdeiras, é aconselhável que o mesmo receba o modificador final por razões de segurança e desempenho.”

Método final: sintaxe

```
<modificador> final <tipo_retorno>
          <nome_metodo> (<lista_parâmetros>){
           //...
   }
```

Método final

```
public class Cpf{

        private long numero;
        private int digito;
        public final boolean valida() {
                // corpo método
                return true;
        }
        // métodos get/set

}
```

Método final

```
public class TesteFinal2{

      public static void main(String args[]) {

        Cpf c = new Cpf();

        if(c.valida())

            System.out.println("Cpf é válido");

    }

}
```

Método final

```
public class Documento{
    //atributos
        public final boolean valida() { ... }
}
public class Cpf extends Documento{
      //atributos
      public boolean valida() {
            // corpo método
     }
}
```

Atributo final

Conhecido como **constante** dos objetos de uma classe

Cuidado! Ao declarar uma variável final é necessário fornecer um valor explícito

Em Java nomenclatura: **todas** letras em **maiúsculas**

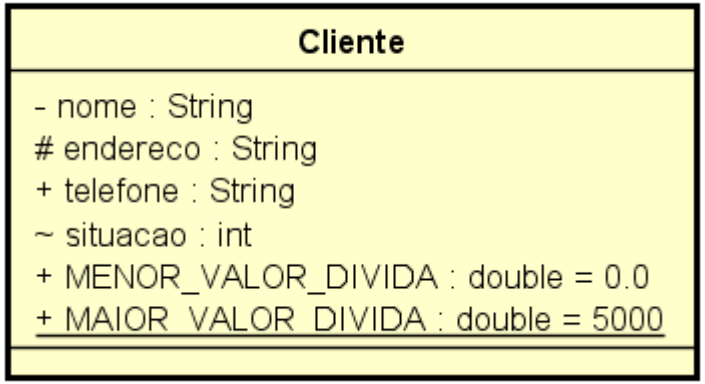
Atributo final: sintaxe

```
<modificador> final <tipo> <nome_variável> = valor;
```

ou:

```
<modificador> static final <tipo> <nome_variável> = valor;
```

Atributo final




```
public class Cliente{

    //…

    public final double MENOR_VALOR_DIVIDA = 0.0;

    public static final double MAIOR_VALOR_DIVIDA = 5000.0;


   //…

}
```

Atributo final

```
public class TesteFinal3{

        public static void main(String args[]) {

          Cliente c = new Cliente();

          System.out.println(c.MENOR_VALOR_DIVIDA);

          System.out.println(Cliente.MAIOR_VALOR_DIVIDA);

    }

}
```

Resumo

Modificador/Elemento	Classe	Método	Atributo
public	✓	✓	✓
private	✗	✓	✓
protected	✗	✓	✓
final	✓	✓	✓

Modificador static

STATIC

Recursos estáticos **pertencem** a uma **classe** e **não** estão associados a uma **instância**

Denominados:

- Atributos estáticos ou variáveis de classe
- Método estáticos ou métodos de classe
- Classe estática – quando é classe interna

Variável de Classe

Apenas uma cópia (classe) para todos as instâncias da classe

Exemplos:

java.lang.Math.E (2.71828...)

java.lang.Math.PI (3.14159...)

Existe alguma constante definida na classe Integer?

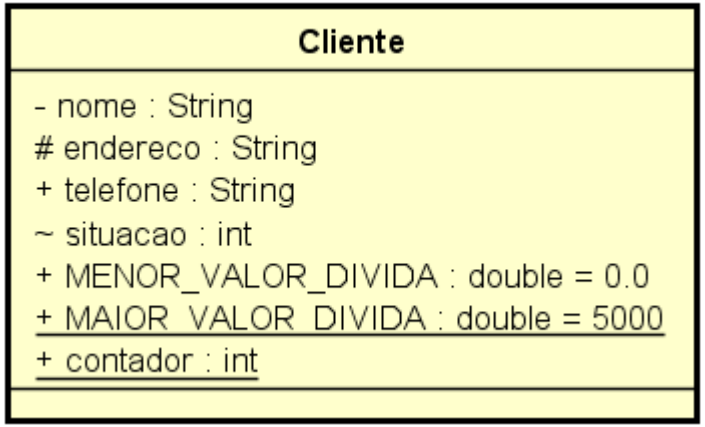
Variável de Classe: sintaxe

```
<modificador> static <tipo> <nome_variável>;
```

ou:

```
<modificador> static final <tipo> <nome_variável>;
```

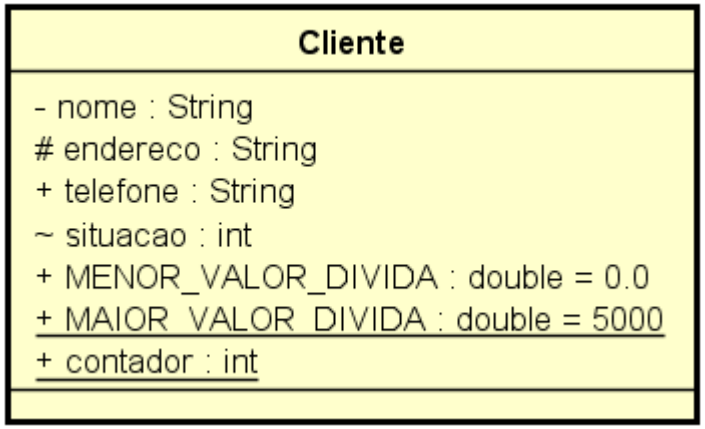
Variável de Classe




```
public class Cliente{
    //…
    public static int contador = 0;
    public static final double MAIOR_VALOR_DIVIDA = 5000.0;
   //…
}
```

Vamos adicionar o contador na classe Cliente

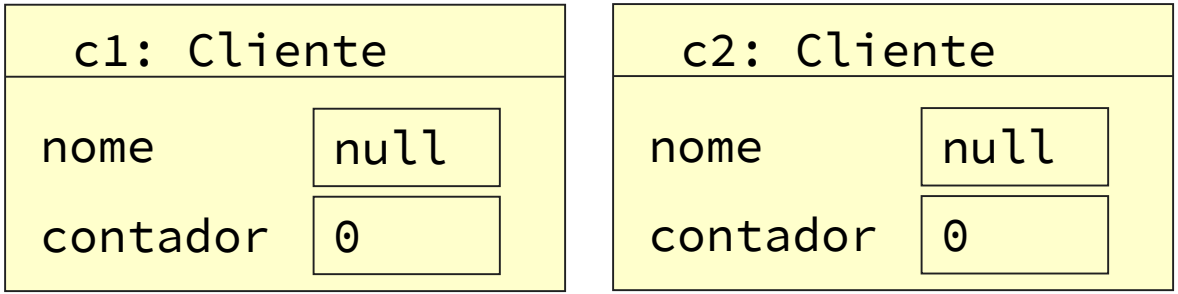
Variável de Classe




```
public class Cliente{
    //…
    public static int contador = 0;
    public Cliente(){ this(null, null, null, 0);}
    public Cliente(String nome, String endereco, String telephone,
int situacao) {
            contador++;
            //outras definições
     }
     //outras definições
}
```

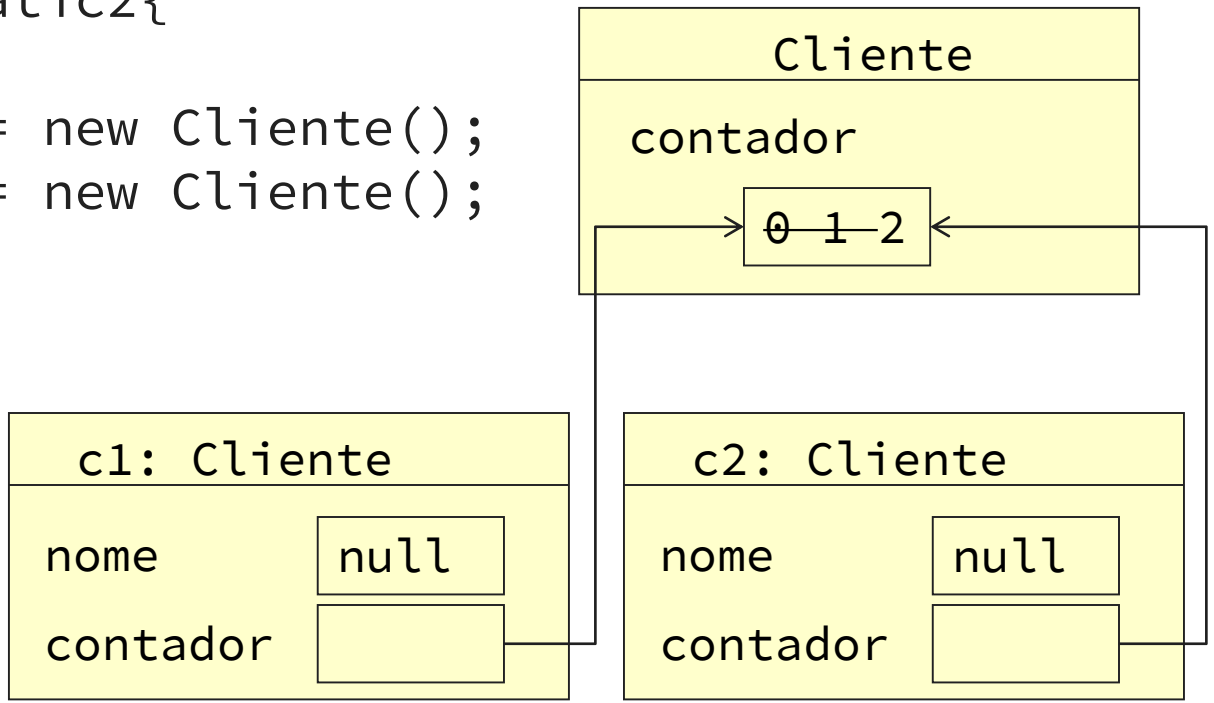
Se contador não fosse Variável de Classe

```
public class TesteStatic1{
  ... main(...){
          Cliente c1 = new Cliente();
          Cliente c2 = new Cliente();
      }
}
```



Se contador não fosse Variável de Classe

```
public class TesteStatic2{
  ... main(...){
          Cliente c1 = new Cliente();
          Cliente c2 = new Cliente();
      }
}
```



Variável de Classe

```
public class TesteStatic2{
  ... main(...){
          Cliente c1 = new Cliente();
          Cliente c2 = new Cliente();
          c1.contador; //ou:
          Cliente.contador;
      }
}
```

Pode ser acessado usando um objeto qualquer ou usando o nome da classe

Método de Classe

Não tem permissão para usar os recursos não estáticos definidos em sua classe:

- Acessar/usar diretamente variáveis de instância
- Chamar diretamente métodos de instância

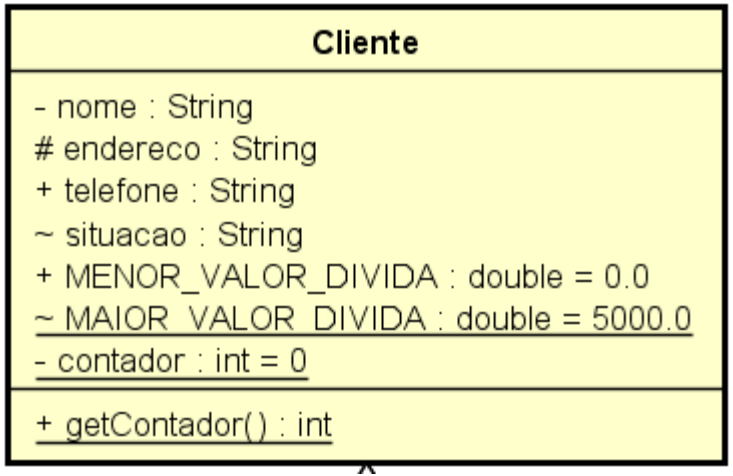
Exemplo: static void main

Método de Classe: sintaxe

```
<modificador> static <tipo_retorno>
                              <nome> (<lista_parâmetros>){
     //...
 }
```

Método de Classe

```
public class Cliente{
    //…
    private static int contador = 0;
    //…
    public static int getContador(){
        return contator;
   }
}
```



Vamos declarar o contador como private e definir o método getContador() na classe Cliente

Método de Classe

```
public class TesteStatic3{
  ... main(...){
          Cliente c1 = new Cliente();
          Cliente c2 = new Cliente();
          int cont1= c1.getContador();
          //ou:
          int cont2 = Cliente.getContador();
    }
}
```

Pode ser acessado usando um objeto qualquer ou usando o nome da classe

Importações estáticas

A partir do J2SDK 5.0 o comando import foi aprimorado para permitir a importação de métodos e variáveis de classe

Exemplo:

```
import static java.lang.System.*;
```

Isso permitirá usar métodos e campos estáticos da classe System sem a necessidade de usar como prefixo o nome da classe:

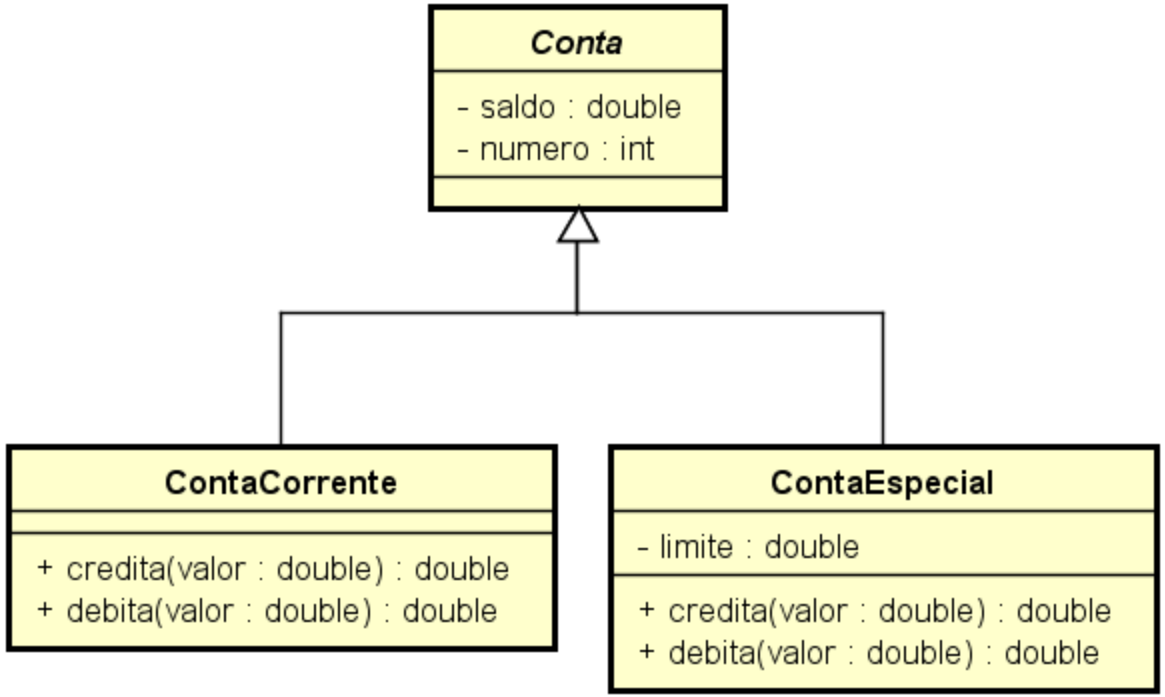
- System.out.println();
- out.println();

Resumo

Modificador/Elemento	Classe	Método	Atributo
public	✓	✓	✓
private	✗	✓	✓
protected	✗	✓	✓
final	✓	✓	✓
static	✓*	✓	✓
static final	✗	✗	✓

*** Usado somente para classes internas**

Exercício


Conta
- saldo : double
- numero : int
ContaCorrente
+ credita(valor : double) : double
+ debita(valor : double) : double
ContaEspecial
- limite : double
+ credita(valor : double) : double
+ debita(valor : double) : double

Modificador abstract

Classe abstrata

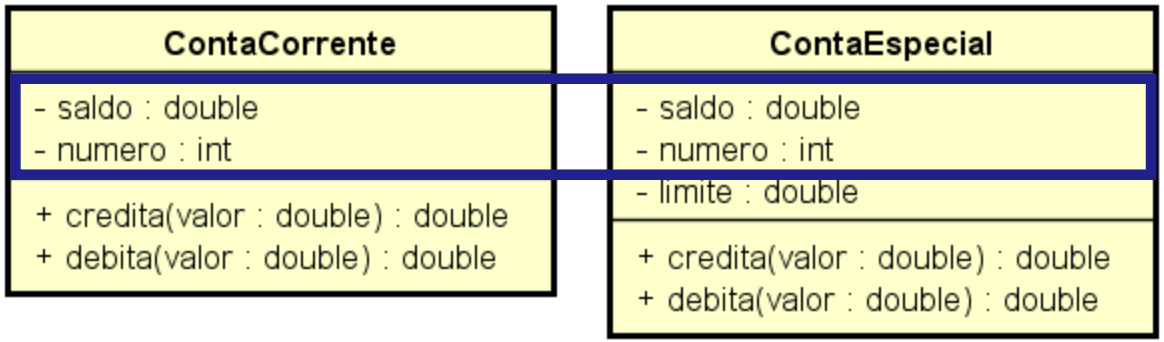
A única finalidade é ser estendida

Incompleta - geralmente, contém métodos abstratos

Métodos **podem** ser definidos nas subclasses

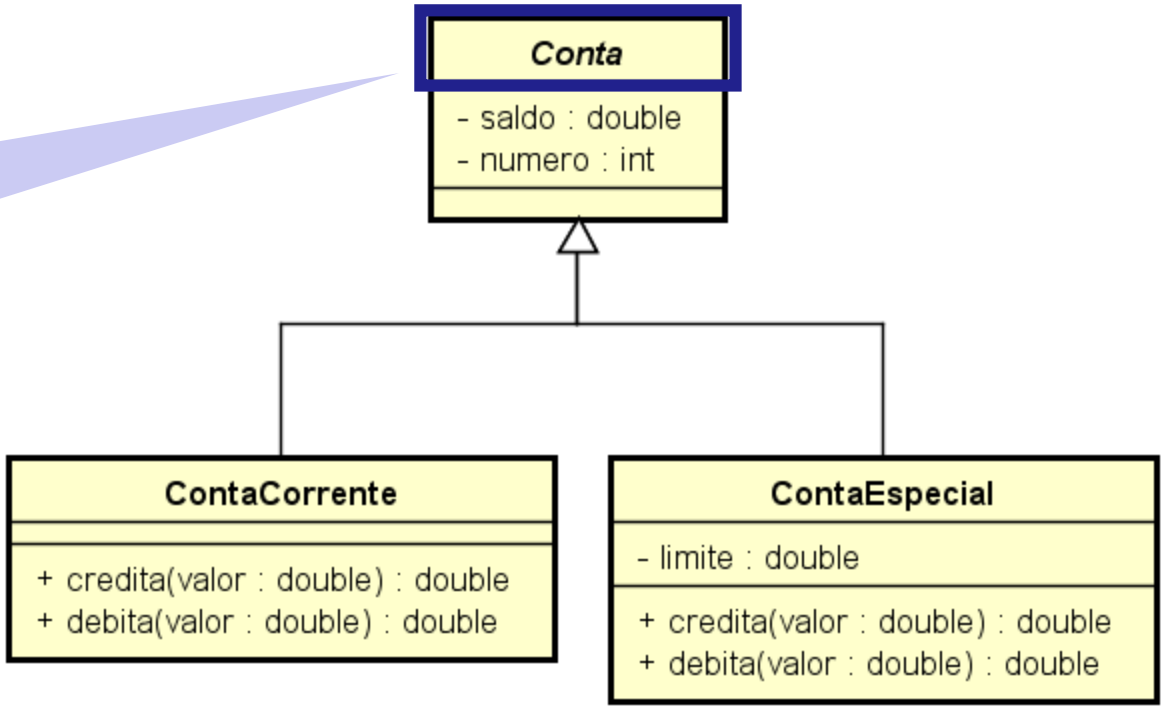
Obs.: se um método for definido como abstrato dentro de uma classe toda a classe deverá ser declarada como abstrata

Classe abstrata



Classe abstrata

Nome em itálico ou com o esteriótipo <<abstract>>



Classe abstrata: sintaxe

```
<modificador> abstract class <nome_classe>{

        //...

}
```

Classe abstrata

```
public abstract class Conta{

    //…

    public Conta(){}

}
```

Classe abstrata

```
public class Teste7{

    public static void main(…){

        Conta c = new Conta();

        c.setSaldo(500.0);

    }

}
```

Método abstrato

Método **declarado**, mas **não** foi **implementado**

Incompleto: falta o corpo

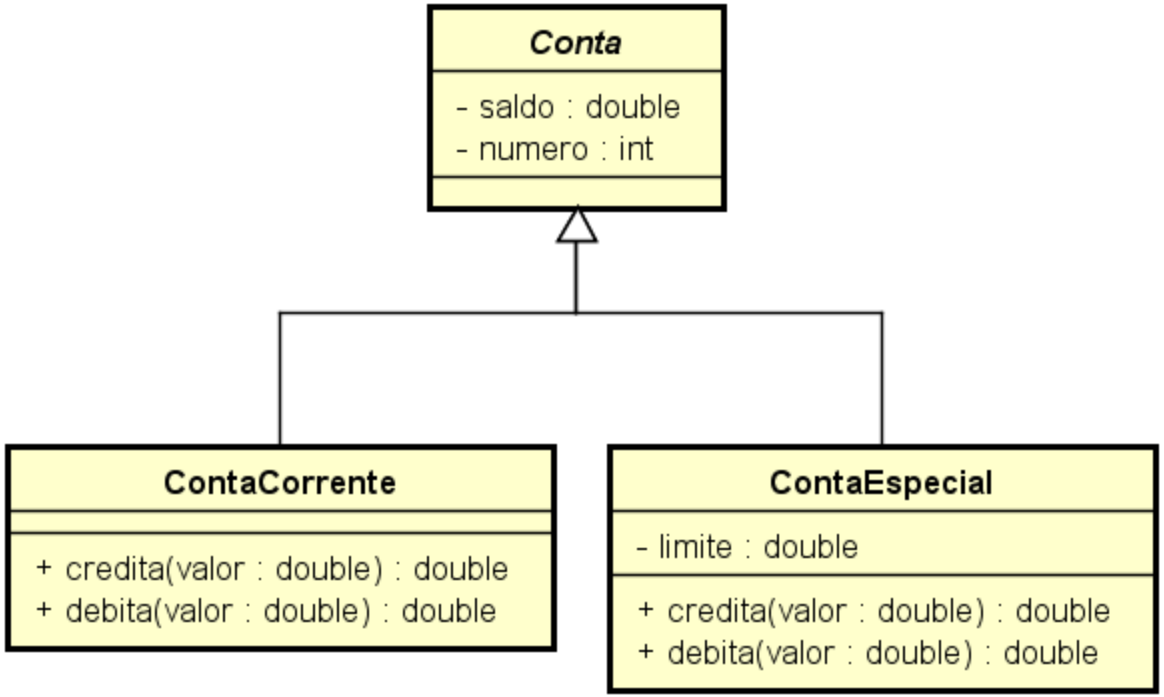
Um método é declarado abstrato quando for significativo para a classe derivada e a implementação não é significativa para a classe base

Subclasse de classe abstrata deve implementar **todos** os **métodos abstratos** da superclasse

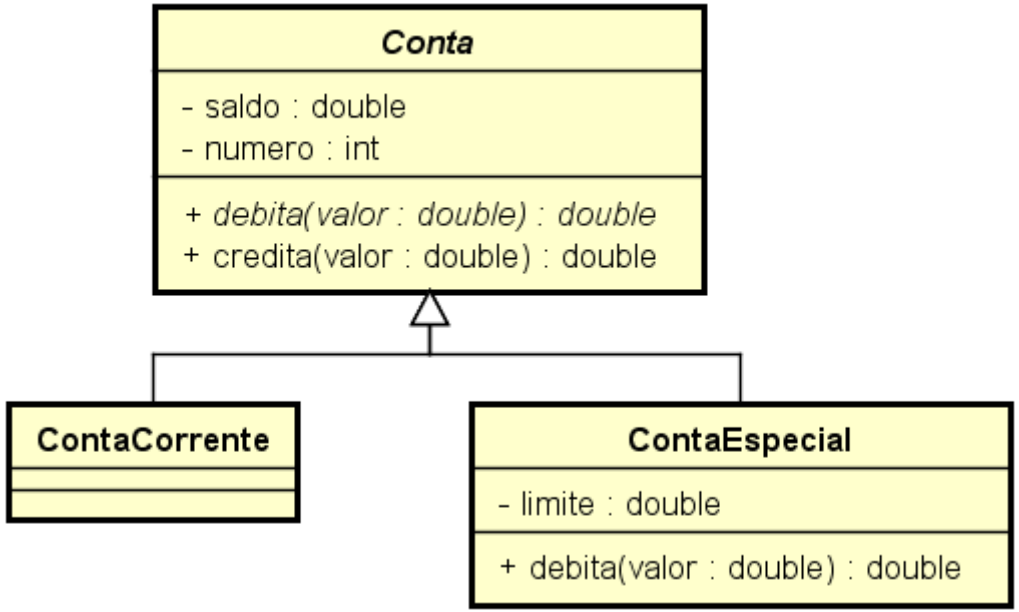
Método abstrato: sintaxe

```
<modificador> abstract <tipo_retorno>
      <nome_metodo> (<lista_parâmetros>);
```

Qual pode ser Método abstrato?



Qual pode ser Método abstrato?



Método abstrato

```
public abstract class Conta{

    private double saldo;
    private int numero;
    public double credita(double valor){
        saldo += valor;
        return saldo;
    }
    public abstract double debita(double valor);

}
```

Um método abstrato não possui implementação, logo usa-se “;” para indicar o término da definição da assinatura do método

Método abstrato

```
public class ContaEspecial extends Conta{
    private double limite;
    public ContaEspecial(){}
    public double debita(double valor){
       if(getSaldo()+limite<=valor)
         setSaldo(getSaldo()-valor);
       return getSaldo();
    }
}
```

Método abstrato

```
public class Teste8{
 public static void main(…){
    ContaEspecial ce = new ContaEspecial(200.0);
    System.out.println(ce.debita(100.0));
 }
}
```

Resumo

Modificador/Elemento	Classe	Método	Atributo
public	✓	✓	✓
private	✗	✓	✓
protected	✗	✓	✓
final	✓	✓	✓
static	✗	✓	✓
static final	✗	✗	✓
abstract	✓	✓	✗
abstract final/abstract	✗	✗	✗
abstract private	✗	✗	✗