Bacharelado em Ciências da Computação

BC1501 – Programação Orientada a Objetos

UFABC

Universidade Federal do ABC

Aula 03

Material Adaptado do
Prof. André G. R. Balan

Aula Passada

- Construtores, Sobrecarga, Palavra **this**

```
class Mesa {
    int numero;
    int lugares;

    public Mesa(int numero) {
        this.numero = numero;
    }
    public Mesa(int numero, int lugares) {
        this.numero = numero;
        this.lugares = lugares;
    }
}
```

Aula Passada

- A classe ArrayList

```
class Pedido {
    int numero;
    Mesa mesa;
    Garcon garcon;
    ArrayList<Prato> pratos;

    public Pedido () {    // construtor
          pratos = new ArrayList<Prato>();
    }

    public void adcionaPrato(Prato prato) {
          pratos.add(prato);
    }
}
```

Aula Passada

- A classe ArrayList

```
class Pedido {
    .
    .
    .

    public void imprime() {
        . . .
       for(Prato p : pratos) {   // Imprime a lista de pratos!!
            System.out.println(p.nome);
        }

        . . .
    }
}
```

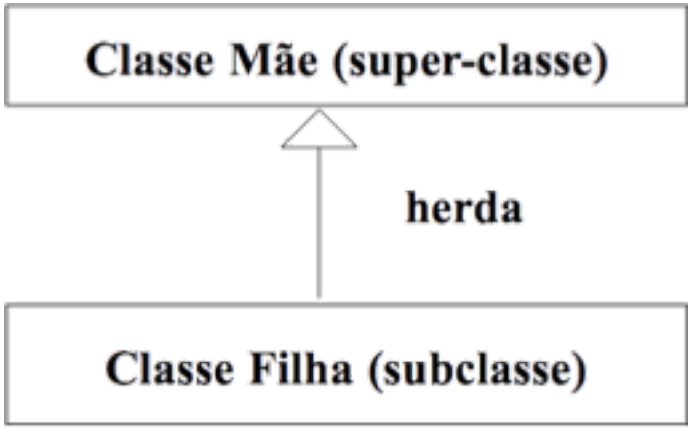
Nesta aula

- **Herança**
- **Polimorfismo**
- **Diagrama de Classes - UML**

Herança

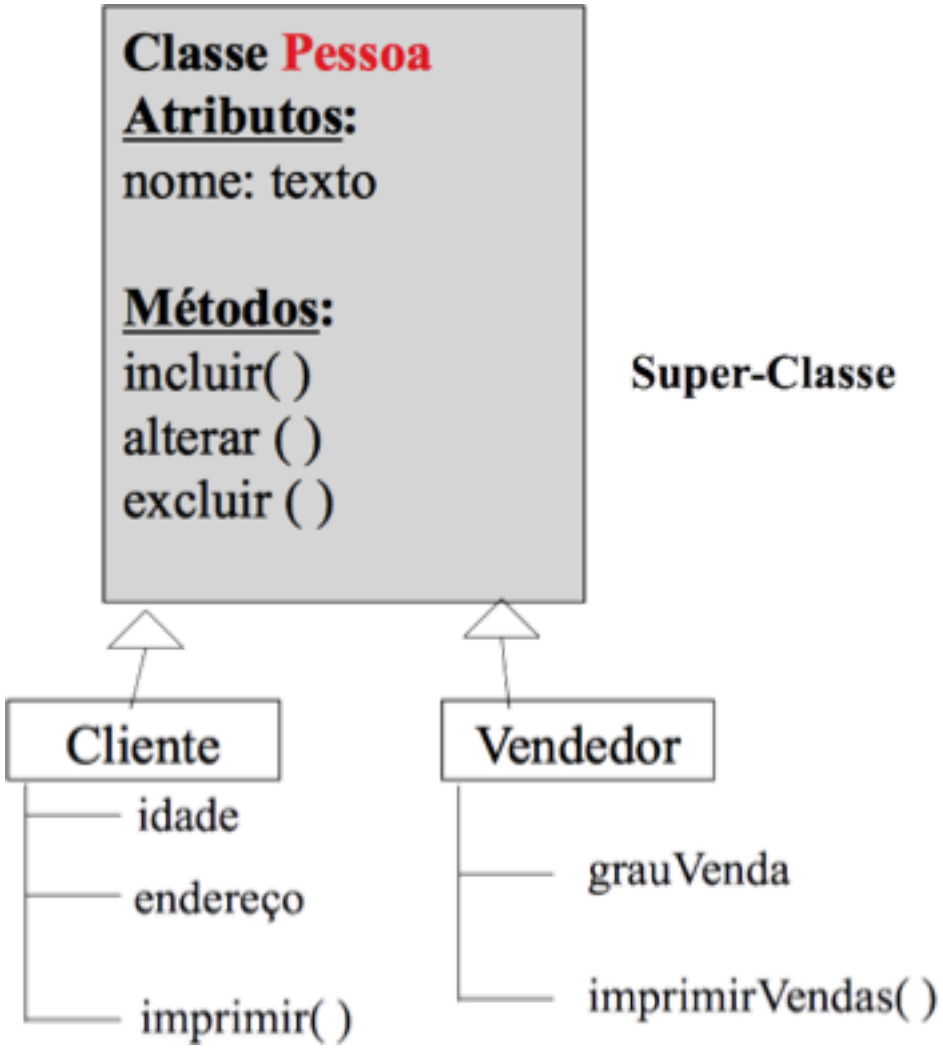
Herança

- Constitui uma relação hierárquica entre classes
  - **Uma classe por ser herdeira, ou filha, de outra classe**



  - **Um classe pode ser mãe (ou pai) de várias outras classes**
- **Quando devemos utilizar a relação de herança?**

Herança



Quando a classe Vendedor é instanciada,

a classe se comportará como Vendedor + Pessoa.

Herança

Quando duas ou mais classes possuem características comuns

Exemplo:

```
class Prato {

   String nome;
   double preço;
   double tempo_preparo;
   .
   .
   .
}
```

```
class Bebida {

   String nome;
   double preço;
   boolean alcoolica;
   int volume_ml;
   .
   .
   .
}
```

Essas classes poderiam ser "irmãs"?

Herança

Quando duas ou mais classes possuem características comuns

Exemplo:

```
class Prato {

   String nome;
   double preço;
   double tempo_preparo;
   .
   .
   .
}
```

```
class Bebida {

   String nome;
   double preço;
   boolean alcoolica;
   int volume_ml;
   .
   .
   .
}
```

Sim! Ambas são coisas que consumimos num restaurante!

Herança

- Isto justifica a criação de uma **classe mãe**

```
class Item_Consumo {

   String nome;
   double preço;

}
```

Herança

- Agora, definimos as classes filhas

```
class Prato extends Item_Consumo {

   double tempo_preparo;
   .
   .
   .
}
```

```
class Bebida extends Item_Consumo {

   boolean alcoolica;
   int volume_ml;
   .
   .
   .
}
```

A propriedades nome e preço estão presentes nas classes filhas por meio de herança!!

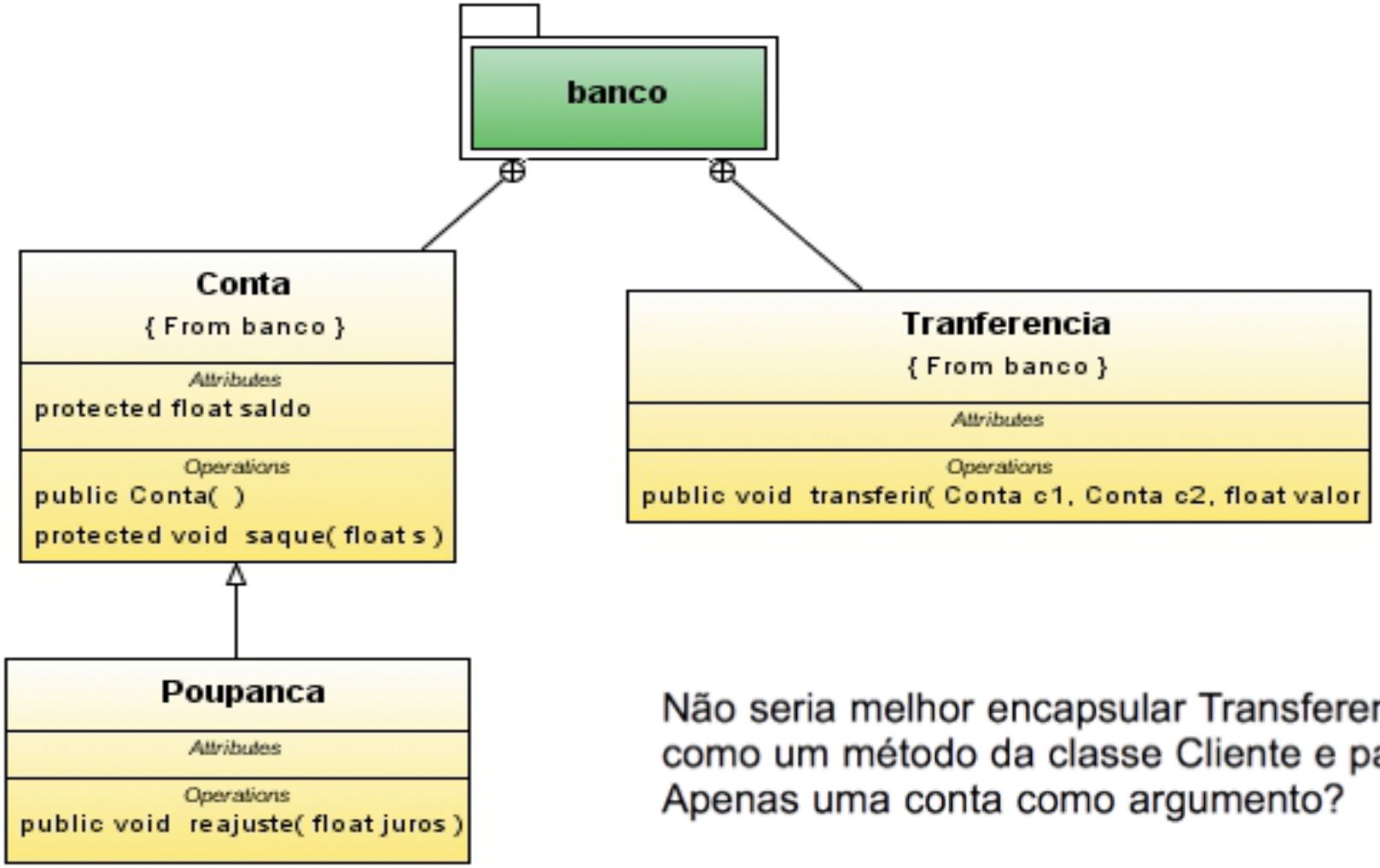
Herança

- Isto ocorre muito frequentemente quando criamos classes

- **Exemplos**
  - Funcionário, Cliente são Pessoas
  - Triângulo, Quadrado, Círculo são Formas_Geométricas
  - Livro, DVD são Itens_Empréstimo em um Biblioteca....

  - ArrayList, LinkedList são filhas da classe List

Herança – Mais exemplos



Não seria melhor encapsular Transferencia como um método da classe Cliente e passar Apenas uma conta como argumento?

Herança – solução em java

```
package banco;

  public class Conta {

      public Conta() { }

      protected float saldo;

      protected void saque(float s) {
          saldo -= s;
      }
  }
```

```
public class Poupanca extends Conta {
       public void reajuste(float juros) {
           saldo *= juros;
       }
   }
```

Herança – solução em java

```
package banco;


  public class Tranferencia {
      public void transferir(Conta c1, Conta c2, float valor) {
          c1.saque(valor);
         c2.saldo += valor;
      }
  }
```

transferir vai retirar **valor** de c1 e depositar em c2

Herança

- **Benefícios**
  - **Evita a duplicação de código** nas classes irmãs. Sendo assim, não é preciso atualizar o código de todas as sub-classes irmãs. Basta atualizar o código da super-classe
  - A Herança, aliada ao **Polimorfismo**, oferece estratégias para programação de **códigos generalizados**

Herança

- **Regras: Exemplo**

```
public class A {

    public int publ;

    protected int prot;

    private int priv;

}
```

Quando criamos um objeto da classe A, podemos acessar somente aquilo que é público

Herança

- **Regras: Exemplo**

```
public class B extends A {


}
```

A classe B herda tudo o que é público e protegido da classe A

É como se copiássemos na classe filha tudo que é público e protegido da classe mãe!

```
public class B extends A {

        public int publ;

        protected int prot;

}
```

Herança

- **Regras: Exemplo**

```
public class B extends A {

        public int publ;

        protected int prot;

}
```

Quando criamos um objeto da classe B, podemos acessar somente aquilo que é público. Neste caso, a **propriedade herdada publ**

A propriedade prot, só poderá ser utilizada nos métodos da classe B!

Herança

- **Regras: Exemplo**

```
public class B extends A {

    . . .

    public void teste() {

        publ = 10;    // ok!

        prot = 10;    // ok!

        priv = 10;    // erro!
                      Não acessível

    }

}
```

Herança

- **Regras: Exemplo**

```
public class C extends B {


}
```

A classe C herda tudo o que é público e protegido da classe B e A!

Herança – resumo

Modificador	Classe	Pacote	Subclasse	Globalmente
Public	sim	sim	sim	sim
Protected	sim	sim	sim	não
Sem Modificador (Padrão)	sim	sim	não	não
Private	sim	não	não	não

Polimorfismo

Polimorfismo

- *Se temos uma variável de referência para objetos da classe A,* ***polimorfismo é a capacidade dessa variável em referenciar normalmente objetos de qualquer classe filha de A***

- ***Ex:***
  - `A vp;`
  - `vp = new B();`

Polimorfismo

- *Observe as classes:*
  - ***Item_Consumo***
    - ***Prato***
    - ***Bebida***

```
Item_Consumo it;
it = new Prato();
it = new Bebida();
```

Polimorfismo

- Sendo assim, se criarmos um ArrayList de Item_Consumo, poderemos colocar tanto Pratos quando Bebidas nesta lista!

```
ArrayList<Item_Consumo> lista;
lista = new ArrayList<Item_Consumo>();

lista.add( new Prato() );
lista.add( new Bebida() );
```

Polimorfismo

- Veja agora a classe pedido

```
class Pedido {
    int numero;
    Mesa mesa;
    Garcon garcon;
    ArrayList<Item_Consumo> itens;

    public Pedido () {    // construtor
        itens = new ArrayList<Item_consumo>();
    }

    public void adcionaItem(Item_Consumo item) {
        itens.add(item);
    }
}
```

Ao invés de termos uma Lista de Pratos e outra Lista de Bebidas, temos uma lista genérica!!

Podemos adicionar tanto objetos da classe Prato quanto Objetos da classe Bebida!

Polimorfismo

- Veja agora a classe pedido

```
class Pedido {
    .
    .
    .
    public void imprime() {
        . . .
       for(Item_Consumo it : itens) {   // Imprime a lista de itens!!
            System.out.println(it.nome);
        }

       . . .
    }
}
```

Polimorfismo envolvendo Métodos

Polimorfismo envolvendo métodos

- Este é um importante exemplo de programação com herança, polimorfismo e métodos de classes

1) *A **classe mãe** define um ou mais **métodos vazios**! (sem implementação)*
2) *As **classes filhas**, definem os mesmos métodos vazios da classe mãe, mas o **método é programado diferente**!*
3) *Com polimorfismo, **a chamada de um método vai assumir o comportamento de acordo com o objeto que o chama!***

Polimorfismo envolvendo métodos

1) *A **classe mãe** define um ou mais **métodos vazios**! (sem implementação)*

```
public abstract class Forma_Geo {
      public int numero;
      public double centro_x, centro_y;
      public abstract void le_dados();
      public abstract void mostra_dados();
      public abstract double area();
      public abstract double perimetro();
}
```

*Para que o método possa ser vazio ele precisa ser definido com a palavra **abstract**, e a classe que o contém também!*

Polimorfismo envolvendo métodos

2) *As **classes filhas**, definem os mesmos métodos vazios da classe mãe, mas, em cada uma, **o método é programado diferente e sem abstract**!*

```
public class Retangulo extends Forma_Geo {
      public double lado1, lado2;

      public void le_dados() {


      }
      public void mostra_dados() {


      }
      . . .
}
```

Polimorfismo envolvendo métodos

2) *As **classes filhas**, definem os mesmos métodos vazios da classe mãe, mas, em cada uma, **o método é programado diferente e sem abstract**!*

```
public class Circulo extends Forma_Geo {
      public double raio;

      public void le_dados() {


      }
      public void mostra_dados() {


      }
      . . .
}
```

Polimorfismo envolvendo métodos

3) *Com polimorfismo,* ***a chamada de um método vai assumir o comportamento de acordo com o objeto!***

```
ArrayList<Forma_Geo> listObjetos = new ArrayList<Forma_Geo>();

listObjetos.add(new Circulo());
listObjetos.add(new Circulo());
listObjetos.add(new Retangulo());

for (Forma_Geo obj : listObjetos)
        obj.le_dados();

for (Forma_Geo obj : listObjetos)
        obj.mostra_dados();
```

Polimorfismo envolvendo métodos

- Com o polimorfismo, uma variável pode apontar para objetos de diversos tipos

- Como sabemos para qual tipo de objeto a variável de referência está apontando em determinado momento?

Polimorfismo envolvendo métodos

- Utilizamos o operador **`instanceof.`** Exemplo:

```
ArrayList<Forma_Geo> listObjetos = new ArrayList<Forma_Geo>();

listObjetos.add(new Circulo());
listObjetos.add(new Circulo());
listObjetos.add(new Retangulo());
int nc = 0;
int nr = 0;

for (Forma_Geo obj : listObjetos)  {
        if (obj instanceof Circulo)    nc++;
        if (obj instanceof Retangulo) nr++;
}
```

Observações

- Não podemos criar objetos de classes abstratas
  - *Por isso o nome "abstrata"*
  - *Elas existem apenas para poder construir programas com códigos generalizados como o que acabamos de ver*
  - *A classe abstrata pode ter métodos não abstratos*

- Quando uma classe é filha de uma classe abstrata, ela é obrigada a implementar todos os métodos abstratos da classe mãe.

Observações

- Existe uma outra importante categoria de classes: as Interfaces!!!
- Uma interface, é como uma classe comum, mas **todos** os seus métodos são implicitamente abstratos!
- As interfaces são utilizadas da mesma forma como vimos no exemplo da classe abstrata.
- Veremos exemplos mais adiante.

Polimorfismo envolvendo métodos

Vamos estudar o programa ObjetosGeometricos feito em Java...

Tarefas

1. Criar outra classe filha da classe Forma_Geo a sua escolha

2. Criar uma entrada no menu para calcular o somatório de todos os perímetros dos objetos cadastrados.