



Programação Orientada à Objetos (POO)

- Instituto Federal de Educação, Ciência e Tecnologia
- de Santa Catarina - Campus São José

- Prof. Glauco Cardozo
- glauco.cardozo@ifsc.edu.br

Programação Orientada à Objetos



- A orientação a objetos é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.
- Orientação a objetos é uma maneira de programar que ajuda na organização e resolve muitos problemas enfrentados pela programação procedural.

Programação Orientada à Objetos



- Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software.
- Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.
- Destre as características que fazem parte de uma classe estão o encapsulamento, a alta coesão e o baixo acoplamento.

Programação Orientada à Objetos



- Encapsulamento:
 - O que começamos a ver nesse capítulo é a ideia de encapsular, isto é, esconder todos os membros de uma classe (como vimos acima), além de esconder como funcionam as rotinas (no caso métodos) do nosso sistema.
 - Encapsular é fundamental para que seu sistema seja suscetível a mudanças: não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada. (veja o caso do método saca)

Programação Orientada à Objetos



- Alta coesão:
 - A coesão está ligada à responsabilidade única da unidade funcional. Demonstra coerência e unidade conceitual no relacionamento com os outros componentes da unidade funcional. Ou seja, um método coeso realiza uma única função conceitual, servindo a apenas um propósito específico.

Programação Orientada à Objetos



- **Baixo Acoplamento**
 - O acoplamento refere-se ao quanto uma unidade funcional depende de outra para funcionar. Uma unidade funcional pode ser um método, função ou mesmo uma classe. Quanto maior a dependência entre as unidades funcionais, mais fortemente acopladas elas estão.

Programação Orientada à Objetos



- Classe
 - Em orientação a objetos, uma classe é uma estrutura que abstrai um conjunto de objetos com características similares. Uma classe define o comportamento de seus objetos através de métodos e os estados possíveis destes objetos através de atributos
- A palavra classe vem da taxonomia da biologia. Todos os seres vivos de uma mesma classe biológica têm uma série de atributos e comportamentos em comum, mas não são iguais, podem variar nos valores desses atributos e como realizam esses comportamentos.

Programação Orientada à Objetos

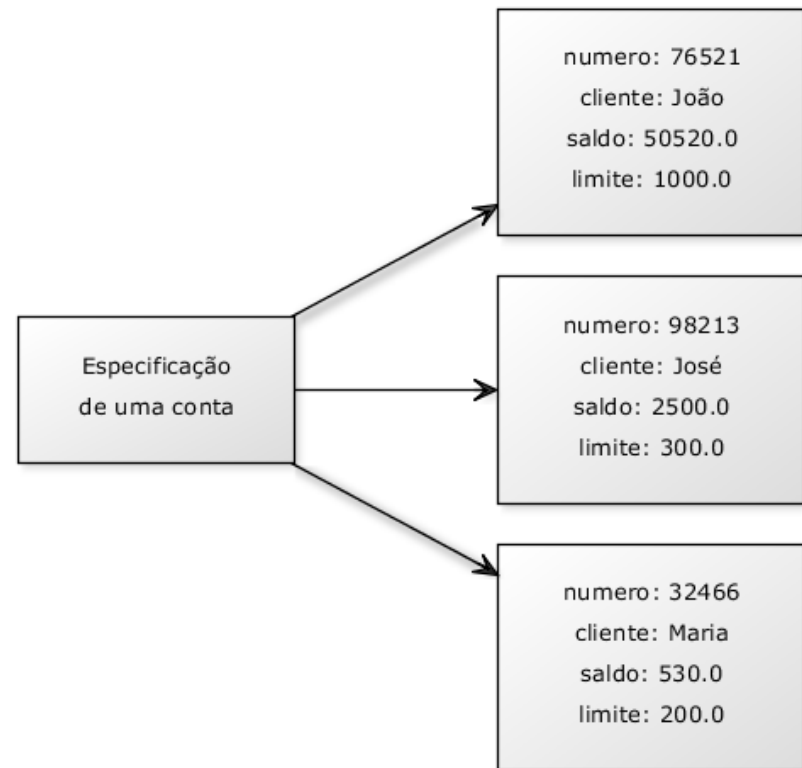


- Objeto ou Instância de uma Classe
 - Um objeto é capaz de armazenar estados através de seus atributos e reagir a mensagens enviadas a ele, assim como se relacionar e enviar mensagens a outros objetos

Programação Orientada à Objetos



- Considere um programa para um banco, é bem fácil perceber que uma entidade extremamente importante para o nosso sistema é a conta.
- Ao projeto da conta, isto é, a definição da conta, damos o nome de classe. Ao que podemos construir a partir desse projeto, as contas de verdade, damos o nome de objetos.



Programação Orientada à Objetos



- O que toda conta tem e é importante para nós?
Quais são seu **atributos**?
 - número da conta
 - nome do dono da conta
 - Saldo
 - Limite
- Atributo são características de uma classe. Basicamente a **estrutura de dados que vai representar a classe.**

Programação Orientada à Objetos



- O que toda conta faz e é importante para nós? Isto é, o que gostaríamos de “pedir à conta”? Quais são os seus **métodos**?
 - saca uma quantidade x
 - deposita uma quantidade x
 - imprime o nome do dono da conta
 - devolve o saldo atual
 - transfere uma quantidade x para uma outra conta y
- Métodos definem as habilidades e/ou **comportamento das classes**,

Programação Orientada à Objetos



- Uma classe em Java

```
class Conta {  
    int numero;  
    String dono;  
    double saldo;  
    double limite;  
  
    // ..  
}
```

Programação Orientada à Objetos



- Métodos

```
class Conta {  
    double salario;  
    // ... outros atributos ...  
  
    void saca(double quantidade) {  
        double novoSaldo = this.saldo - quantidade;  
        this.saldo = novoSaldo;  
    }  
  
    void deposita(double quantidade) {  
        this.saldo += quantidade;  
    }  
}
```

Programação Orientada à Objetos



- Métodos com retorno

- Um método pode retornar um valor para o código que o chamou.

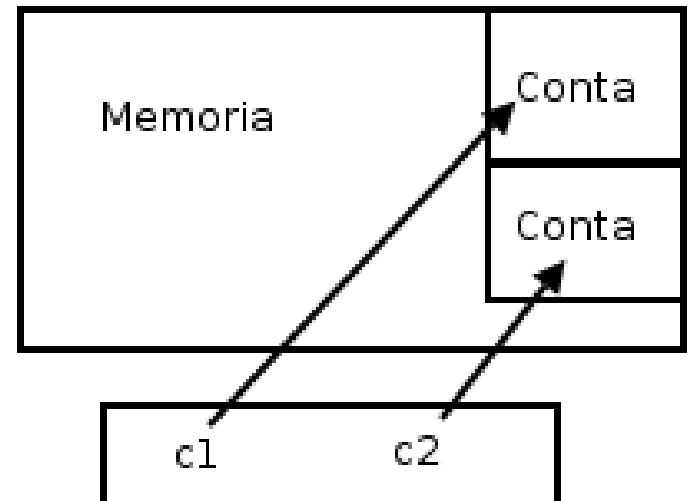
```
class Conta {  
    // ... outros métodos e atributos ...  
  
    boolean saca(double valor) {  
        if (this.saldo < valor) {  
            return false;  
        }  
        else {  
            this.saldo = this.saldo - valor;  
            return true;  
        }  
    }  
}
```

Programação Orientada à Objetos



- Objetos são acessados por referências
 - Quando declaramos uma variável para associar a um objeto, na verdade, essa variável não guarda o objeto, e sim uma maneira de acessá-lo, chamada de referência.

```
public static void main(String args[]) {  
    Conta c1;  
    c1 = new Conta();  
  
    Conta c2;  
    c2 = new Conta();  
}
```



Programação Orientada à Objetos



- Objetos são acessados por referências

```
class TestaReferencias {  
    public static void main(String args[]) {  
        Conta c1 = new Conta();  
        c1.deposita(100);  
  
        Conta c2 = c1; // linha importante!  
        c2.deposita(200);  
  
        System.out.println(c1.saldo);  
        System.out.println(c2.saldo);  
    }  
}
```


Programação Orientada à Objetos



- O método transfere()

```
class Conta {  
  
    // atributos e métodos...  
  
    void transfere(Conta destino, double valor) {  
        this.saldo = this.saldo - valor;  
        destino.saldo = destino.saldo + valor;  
    }  
}
```

Programação Orientada à Objetos



- O método transfere()

```
boolean transfere(Conta destino, double valor) {  
    boolean retirou = this.saca(valor);  
    if (retirou == false) {  
        // não deu pra sacar!  
        return false;  
    }  
    else {  
        destino.deposita(valor);  
        return true;  
    }  
}
```

Programação Orientada à Objetos



- Atributos

```
class Conta {  
    int numero = 1234;  
    String dono = "Duke";  
    String cpf = "123.456.789-10";  
    double saldo = 1000;  
    double limite = 1000;  
}
```

Programação Orientada à Objetos



- Atributos

```
class Cliente {  
    String nome;  
    String sobrenome;  
    String cpf;  
}
```

```
class Conta {  
    int numero;  
    double saldo;  
    double limite;  
    Cliente titular;  
    // ..  
}
```

Programação Orientada à Objetos



- Testando

```
class Teste {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
        Cliente c = new Cliente();  
        minhaConta.titular = c;  
        // ...  
    }  
}
```

```
    minhaConta.titular.nome = "Duke";
```

Programação Orientada à Objetos



- Testando

```
class Teste {  
    public static void main(String[] args) {  
        Conta minhaConta = new Conta();  
  
        minhaConta.titular.nome = "Manoel";  
        // ...  
    }  
}
```

minhaConta



numero	
saldo	
limite	
Cliente	null

Programação Orientada à Objetos



- Testando

```
class Conta {  
    int numero;  
    double saldo;  
    double limite;  
    Cliente titular = new Cliente();  
  
}
```



Programação Orientada à Objetos

Aula 2

Programação Orientada à Objetos



- Encapsulamento
 - Encapsular é fundamental para que seu sistema seja suscetível a mudanças: não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada.
 - Programando voltado para a interface e não para a implementação

Programação Orientada à Objetos



- Modificador de acesso
 - Cada classe é responsável por controlar seus atributos, portanto ela deve julgar se aquele novo valor é válido ou não! Esta validação não deve ser controlada por quem está usando a classe e sim por ela mesma, centralizando essa responsabilidade e facilitando futuras mudanças no sistema.

```
class Conta {  
    private double saldo;  
    private double limite;  
    // ...  
}
```

Marcando um atributo como **privado**, fechamos o acesso ao mesmo em relação a todas as outras classes

Programação Orientada à Objetos



- Modificador de acesso

```
class Conta {  
    private double saldo;  
    private double limite;  
    // ...  
}
```

```
class TestaAcessoDireto {  
    public static void main(String args[]) {  
        Conta minhaConta = new Conta();  
        //não compila! você não pode acessar o atributo  
        minhaConta.saldo = 1000;  
    }  
}
```

Programação Orientada à Objetos



- **Getters e Setters**

- O modificador `private` faz com que ninguém consiga modificar, nem mesmo ler, o atributo em questão
- Precisamos ter uma forma de acessar os atributos

```
private double saldo;  
private double limite;  
private Cliente titular;
```

```
public double getSaldo() {  
    return this.saldo;  
}
```

```
public void setSaldo(double saldo) {  
    this.saldo = saldo;  
}
```

Programação Orientada à Objetos



- Construtores

- Quando usamos a palavra chave new, estamos construindo um objeto. Sempre quando o new é chamado, ele executa o construtor da classe. O construtor da classe é um bloco declarado com o mesmo nome que a classe:

```
class Conta {
    int numero;
    Cliente titular;
    double saldo;
    double limite;

    // construtor
    Conta() {
        System.out.println("Construindo uma conta.");
    }

    // ..
}
```

Programação Orientada à Objetos



- Construtores

- É utilizado para inicializar uma classe.

```
class Conta {  
    int numero;  
    Cliente titular;  
    double saldo;  
    double limite;  
  
    // construtor  
    Conta(Cliente titular) {  
        this.titular = titular;  
    }  
}
```

Programação Orientada à Objetos



- Atributos de classe
 - Nosso banco também quer controlar a quantidade de contas existentes no sistema. Como poderíamos fazer isto

```
Conta c = new Conta();
```

```
totalDeContas = totalDeContas + 1;
```

```
private int totalDeContas;
```

```
//...
```

```
Conta() {
```

```
    this.totalDeContas = this.totalDeContas + 1;
```

```
}
```

O atributo é de cada objeto.

Programação Orientada à Objetos



- Atributos de classe

- Seria interessante então, que essa variável fosse única, compartilhada por todos os objetos dessa classe. Dessa maneira, quando mudasse através de um objeto, o outro enxergaria o mesmo valor. Para fazer isso em java, declaramos a variável como **static**.

```
private static int totalDeContas;  
//...
```

```
Conta() {  
    Conta.totalDeContas = Conta.totalDeContas + 1;  
}
```


Programação Orientada à Objetos



- Método de classe

- A ideia aqui é a mesma, transformar esse método que todo objeto conta tem em um método de toda a classe. Usamos a palavra `static` de novo, mudando o método anterior.

```
public static int getTotalDeContas() {  
    return Conta.totalDeContas;  
}
```



Programação Orientada à Objetos



- Herança

```
class Funcionario {  
    String nome;  
    String cpf;  
    double salario;  
    // métodos devem vir aqui  
}
```

Programação Orientada à Objetos



- Herança

```
class Gerente {
    String nome;
    String cpf;
    double salario;
    int senha;
    int numeroDeFuncionariosGerenciados;

    public boolean autentica(int senha) {
        if (this.senha == senha) {
            System.out.println("Acesso Permitido!");
            return true;
        } else {
            System.out.println("Acesso Negado!");
            return false;
        }
    }
}
```

Programação Orientada à Objetos



- Herança
 - Existe um jeito, em Java, de relacionarmos uma classe de tal maneira que uma delas **herda** tudo que a outra tem. Isto é uma relação de classe mãe e classe filha. No nosso caso, gostaríamos de fazer com que o Gerente tivesse tudo que um Funcionario tem, gostaríamos que ela fosse uma **extensão** de Funcionario. Fazemos isto através da palavra chave extends.

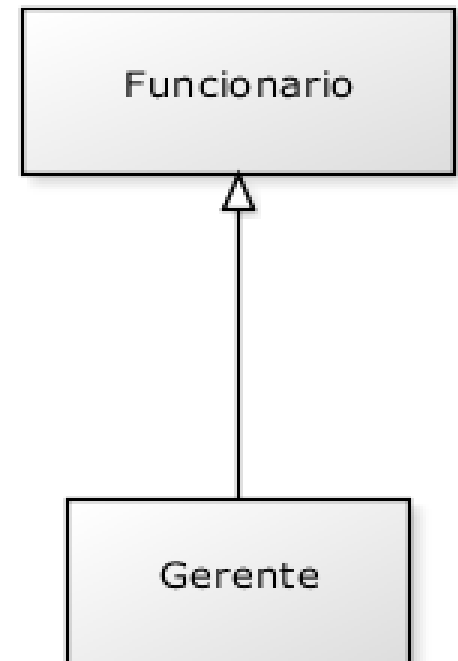
Programação Orientada à Objetos



- Herança

```
class Gerente extends Funcionario {  
    int senha;  
    int numeroDeFuncionariosGerenciados;  
}
```

Dizemos que a classe Gerente **herda** todos os atributos e métodos da classe mãe, no nosso caso, a Funcionario. Para ser mais preciso, ela também herda os atributos e métodos privados, porém não consegue acessá-los diretamente. Para acessar um membro privado na filha indiretamente, seria necessário que a mãe expusesse um outro método visível que invocasse esse atributo ou método privado.



Programação Orientada à Objetos



- Herança

Super e Sub classe

A nomenclatura mais encontrada é que Funcionario é a **superclasse** de Gerente, e Gerente é a **subclasse** de Funcionario. Dizemos também que todo Gerente é **um** Funcionário. Outra forma é dizer que Funcionario é classe **mãe** de Gerente e Gerente é classe **filha** de Funcionario.

Programação Orientada à Objetos



- Herança

```
class Funcionario {  
    protected String nome;  
    protected String cpf;  
    protected double salario;  
    // métodos devem vir aqui  
}
```


Programação Orientada à Objetos



- Reescrita de método

Todo fim de ano, os funcionários do nosso banco recebem uma bonificação. Os funcionários comuns recebem 10% do valor do salário e os gerentes, 15%.

```
class Funcionario {  
    protected String nome;  
    protected String cpf;  
    protected double salario;  
  
    public double getBonificacao() {  
        return this.salario * 0.10;  
    }  
    // métodos  
}
```

Programação Orientada à Objetos



- Reescrita de método

Todo fim de ano, os funcionários do nosso banco recebem uma bonificação. Os funcionários comuns recebem 10% do valor do salário e os gerentes, 15%.

```
class Gerente extends Funcionario {  
    int senha;  
    int numeroDeFuncionariosGerenciados;  
  
    public double getBonificacao() {  
        return this.salario * 0.15;  
    }  
    // ...  
}
```

Programação Orientada à Objetos



- Reescrita de método

Imagine que para calcular a bonificação de um Gerente devemos fazer igual ao cálculo de um Funcionario porem adicionando R\$ 1000. Poderíamos fazer assim:

```
class Gerente extends Funcionario {  
    int senha;  
    int numeroDeFuncionariosGerenciados;  
  
    public double getBonificacao() {  
        return this.salario * 0.10 + 1000;  
    }  
}
```

Programação Orientada à Objetos



- Reescrita de método

Imagine que para calcular a bonificação de um Gerente devemos fazer igual ao cálculo de um Funcionario porem adicionando R\$ 1000. Poderíamos fazer assim:

```
class Gerente extends Funcionario {  
    int senha;  
    int numeroDeFuncionariosGerenciados;  
  
    public double getBonificacao() {  
        return super.getBonificacao() + 1000;  
    }  
}
```