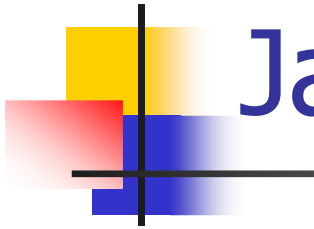
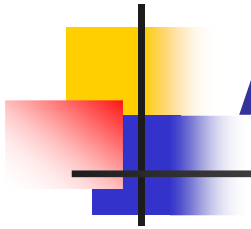


Java Standard Edition (JSE)

Capítulo 06. Herança, Reescrita e Polimorfismo

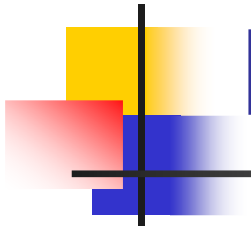
Esp. Márcio Palheta

MSN: marcio.palheta@hotmail.com



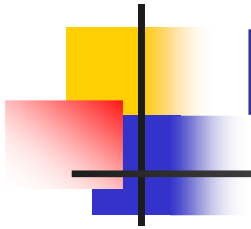
Agenda

- Revisão da aula anterior;
- Motivação – Ame o que faz;
- Herança;
- Reescrita de métodos;
- Polimorfismo;
- Exercícios de fixação



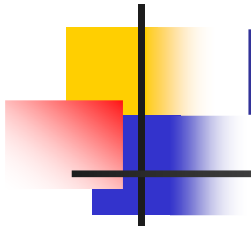
Revisão

- Classe, objeto, atributos e métodos;
- Controle de acesso a atributos e métodos;
- Encapsulamento;
- Getters e Setters;
- Construtores;
- Dúvidas ?



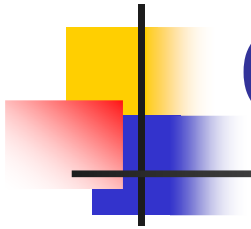
Motivação – Ame o que faz

- Dedique-se ao trabalho;
- Esteja pronto para o mercado;
- Qualifique-se;
- Parabéns! Estamos chegando ao final da nossa disciplina;
- Se você está aqui, é por que pretende se tornar um bom programador, no mínimo;
- Video:
  - 06.01 Eu sou uma anta



Novos recursos a aprender

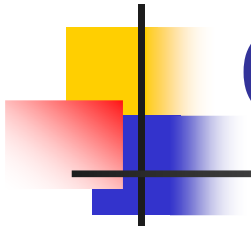
- O que é herança e quando utilizá-la;
- Reutilizar código escrito anteriormente;
- Criar classes filhas e reescrever métodos;
- Usar todo o poder que o polimorfismo propicia;



Cenário 01

- Como em outras empresas, nosso banco possui funcionários com: nome, cpf e salário:

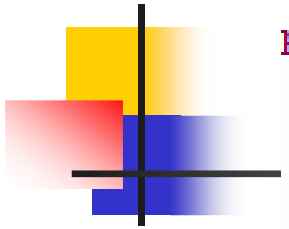
```
public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;

    public String getCpf() {..
    public void setCpf(String cpf) {..
    public String getNome() {..
    public void setNome(String nome) {..
    public double getSalario() {..
    public void setSalario(double salario) {..
}
```



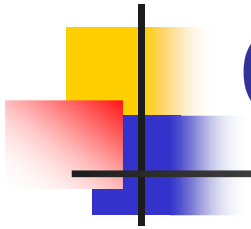
Cenário 01

- O banco possui, também, os gerentes que, além dos mesmos atributos e métodos de um funcionário, possui:
  - uma senha numérica para acesso aos sistemas internos da empresa;
  - Um método booleano autenticar que informa se a senha informada está correta;

Classe Gerente

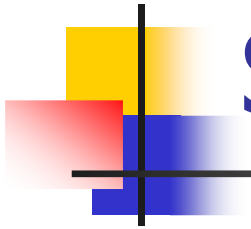


```
public class Gerente {
    private String nome;
    private String cpf;
    private double salario;
    private int senha;

    public boolean autenticar(int senha){
        return (this.senha == senha);
    }
    public int getSenha() {
        return senha;
    }
    public void setSenha(int senha) {
        this.senha = senha;
    }
    public String getCpf() {..
    public void setCpf(String cpf) {..
    public String getNome() {..
    public void setNome(String nome) {..
    public double getSalario() {..
    public void setSalario(double salario) {..
}
```



Considerações

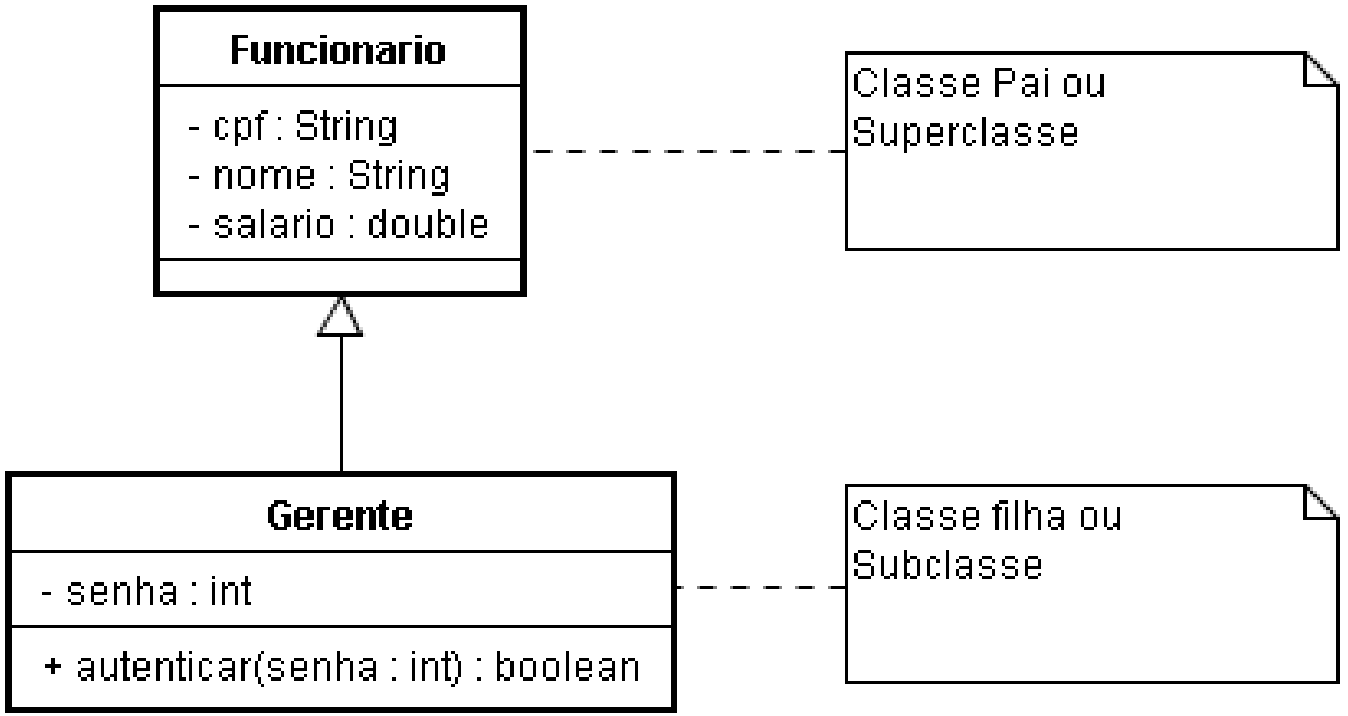
- Os atributos e métodos das classe são os mesmos, com exceção dos componentes que pertencem apenas à classe Gerente;
- Vamos repetir o código para atributos e métodos iguais?
- A alteração do método setCpf(), por exemplo tem que ser feita em duas classes distintas;



Solução: Herança

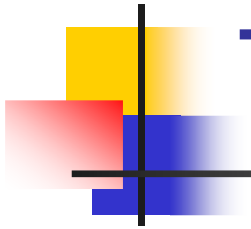
- No java, podemos criar classe a Funcionario com todos os componentes comuns;
- Em seguida, podemos dizer que a classe Gerente é uma extensão da classe Funcionario;
- Com isso, todos os componentes declarados em Funcionario, poderão ser utilizados por Gerente;

Herança – Representação Gráfica UML



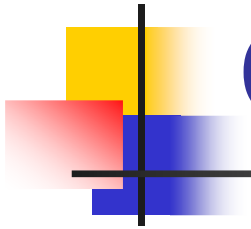
Alteração da classe Gerente

```
public class Gerente extends Funcionario{
    private int senha;

    public boolean autenticar(int senha){
        return (this.senha == senha);
    }
    public int getSenha() {
        return senha;
    }
    public void setSenha(int senha) {
        this.senha = senha;
    }
}
```



Teste da classe Gerente

- class TestaGerente {
- public static void main(String[] args) {
  - Gerente gerente = new Gerente();
  - gerente.setNome("João da Silva");
  - gerente.setSenha(4231);
  - }
- }
- Dizemos que a classe Gerente **herda** todos os atributos e métodos da classe mãe, no nosso caso, Funcionario, embora não consiga acessar diretamente os componentes privados;

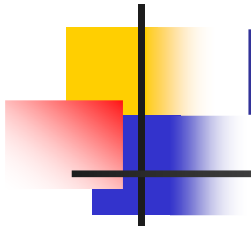


Cenário 02

- Todo fim de ano, os funcionários do banco recebem bonificação.
- Os funcionários comuns recebem 10% do valor do salário e os gerentes, 15%;
- Vamos criar o método getBonificacao(), na classe Funcinoario, que retorna um double com o valor do bônus;
- Como Gerente é um extends de Funcionário, herdará o novo método;

Atualização da classe Funcionario

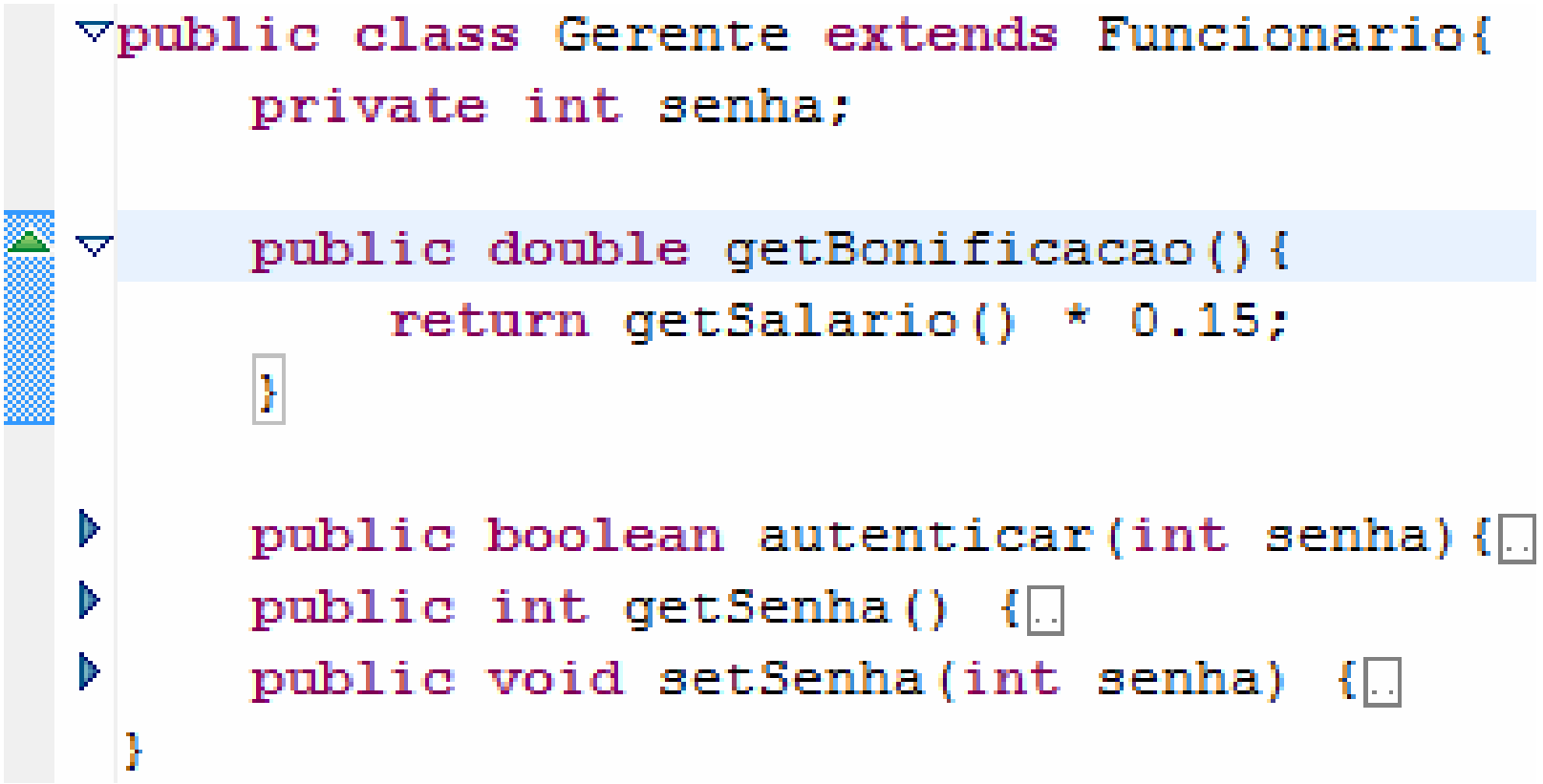
```
public class Funcionario {
    private String nome;
    private String cpf;
    private double salario;

    public double getBonificacao(){
        return this.salario * 1.2;
    }

    public String getNome() {
    public void setNome(String nome) {
    public String getCpf() {
    public void setCpf(String cpf) {
    public double getSalario() {
    public void setSalario(double salario) {
}
```



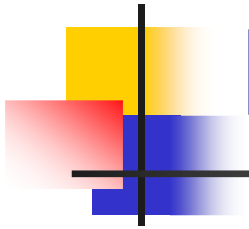
Reescrita de método

- Diferente de Funcionario, o bônus de gerente equivale a 15% do valor do salário;
- Para resolver o problema, na classe Gerente, vamos reescrever o método getBonificacao, herdado da classe Funcionario;
- Para isso, repetimos a assinatura do método e alteramos sua implementação

Reescrita – getBonificacao()



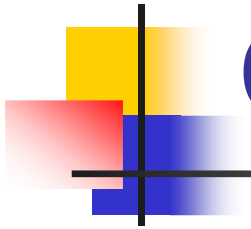
```
public class Gerente extends Funcionario{
    private int senha;

    public double getBonificacao(){
        return getSalario() * 0.15;
    }

    public boolean autenticar(int senha){..
    public int getSenha() {..
    public void setSenha(int senha) {..
}
```



Exercício 01

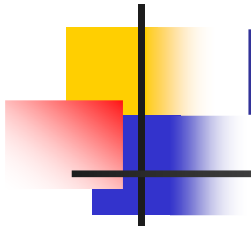
- O que será impresso no código abaixo?

```
public static void main(String[] args) {
    Funcionario orelha = new Funcionario();
    orelha.setSalario(1300);
    Gerente chefao = new Gerente();
    chefao.setSalario(5000);
    System.out.println("Bônus do funcionário: " +
                       orelha.getBonificacao());
    System.out.println("Bônus do gerente: " +
                       chefao.getBonificacao());
}
```



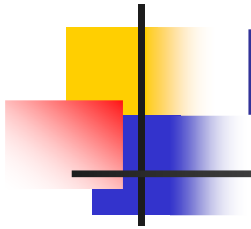
Cenário 03

- Na herança, vimos que todo Gerente é um Funcionario, pois é uma extensão deste.
- Podemos nos referir a um Gerente como sendo um Funcionario.
- Se alguém precisa falar com um Funcionario do banco, pode falar com um Gerente!
- Por quê? O Gerente **é um** Funcionario.
  - Gerente gerente = new Gerente();
  - Funcionario funcionario = gerente;
  - funcionario.setSalario(5000.0);



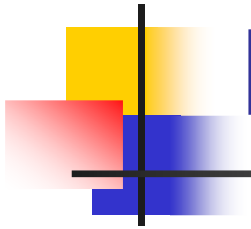
Polimorfismo

- É a capacidade de um objeto ser referenciado de diversas formas;
- Se incluirmos a linha a baixo no código anterior, o que será retornado?
  - double bonus = funcionario.getBonificacao();
- bonus == 500 ou 750?
- A invocação do método é feita em tempo de execução, utilizando a classe real do objeto. bonus == 750



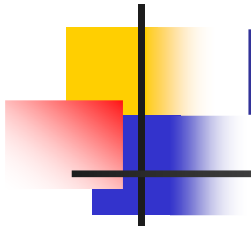
Exercício 02

- Vamos controlar o total de bônus pago;
- Criaremos uma classe ControleBonificacao que vai calcular o total de bônus pago;
- O método de controle de bonus deve receber um objeto Funcionario, e atualizar o total pago, de acordo com getBonificacao();
- Crie uma classe teste para ler os dados de N Funcionarios: salário e tipo(F ou G)e atualizar o total de bônus;
- Imprima o total de bônus do banco;



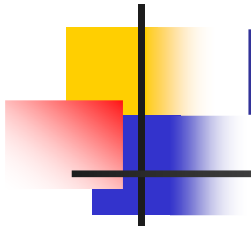
Exercício 03

- Vamos criar uma classe Conta, que possua um saldo, e os métodos para pegar saldo, depositar, e sacar;
- Adicione um método à classe Conta, que atualiza essa conta de acordo com uma taxa percentual fornecida;
- Crie duas subclasses da classe Conta: ContaCorrente e ContaPoupanca;
- Sobrescreva os métodos depositar e sacar de ContaCorrente, para cobrar taxa de serviço de 1% do valor da transação;



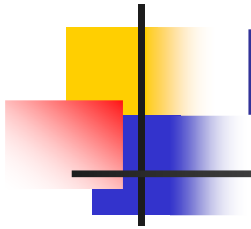
Exercício 04

- Crie uma classe Banco que possui um array de Conta.
- Em array de Conta você pode colocar ContaCorrente e ContaPoupanca.
  - Crie um método void adiciona(Conta conta);
  - Conta pegaConta(int x);
  - int pegaTotalDeContas();
- Faça com que seu método main crie diversas contas, insira-as no Banco e depois, com um for, percorra todas as contas do Banco para informar o número e o saldo;



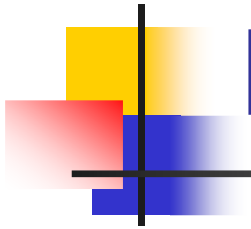
Exercício 05

- Crie uma classe chamada **Pessoa**. Uma pessoa possui um nome e uma idade.
- Crie 2 construtores: 1 que recebe o nome e a idade como parâmetros de entrada e um que não recebe parâmetros e inicializa os atributos com um valor padrão (“indefinido” para Strings e 0 para inteiros).
- Crie os métodos de acesso para os atributos (GET e SET).



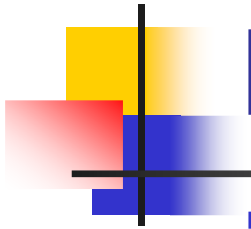
Exercício 06

- Crie uma classe **Amigo**, que herda **Pessoa**, e possui uma data de aniversário;
- Crie um construtor que não recebe parâmetros de entrada, e inicializa o atributo com um valor padrão;
- Crie os métodos de acesso para o atributo data de nascimento.



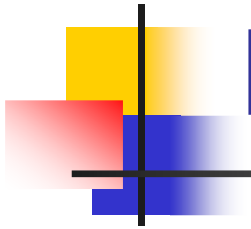
Exercício 07

- Crie uma classe **Conhecido**, que herda **Pessoa**, e possui um email;
- Crie um construtor que não recebe parâmetros de entrada, e inicializa o email com um valor padrão ;
- Crie os métodos de acesso para este atributo.



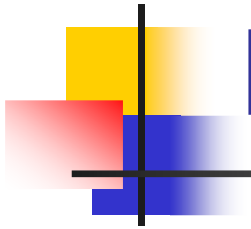
Exercício 08 - Agenda

- Crie agora, uma classe **Agenda**, que possui pessoas (em um ArrayList) e dois atributos que controlam: a quantidade de amigos e a quantidade de conhecidos.
- crie um construtor que recebe por parâmetro a quantidade inicial de pessoas que a agenda terá, e inicializa o ArrayList de **Pessoa**. Neste construtor, inicialize todas as posições do array criando *ALEATORIAMENTE* um **Conhecido** ou um **Amigo;**
- Utilize o comando: 1 + (int) (Math.random() * 2 para sortear valores entre 1 e 2. Se o valor encontrado for 1, crie um **Amigo**. Se o valor encontrado for 2, crie um **Conhecido**).
- Crie os métodos *GET* para todos os atributos da classe **Agenda**.
- Crie um método chamado *addInformacoes*, que não recebe parâmetros de entrada. Para cada **Pessoa** na agenda, peça para o usuário digitar as informações cabíveis para cada tipo de **Pessoa**, e acesse os métodos *SET* para atribuir as informações.
- Crie um método chamado *imprimeAniversários*, que imprime os aniversários de todos os amigos que estão armazenados na agenda.
- Crie um método chamado *imprimeEmail*, que imprime os e-mails de todos os conhecidos que estão armazenados na agenda.



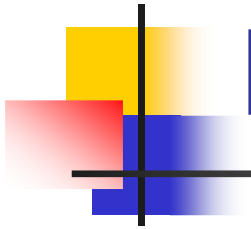
Exercício 09 - AgendaTeste

- Crie uma classe de teste para a **Agenda**.
- Peça para o usuário informar quantas pessoas ele deseja colocar na agenda, e crie uma **Agenda** com esta informação.
- Imprima na tela a quantidade de amigos e de conhecidos na agenda.
- Adicione informações à agenda.
- Imprima todos os aniversários dos amigos presentes na agenda.
- Imprima todos os e-mails dos conhecidos armazenados na agenda.



Bibliografia

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



Referências WEB

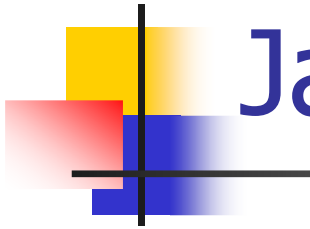
- SUN: www.java.sun.com

Fóruns e listas:

- Javaranch: www.javaranch.com
- GUJ: www.guj.com.br

Apostilas:

- Argonavis: www.argonavis.com.br
- Caelum: www.caelum.com.br



Java Standard Edition (JSE)

Capítulo 06. Herança, Reescrita e Polimorfismo

Esp. Márcio Palheta

MSN: marcio.palheta@hotmail.com