

# Desenvolvimento de Software

Juliana Costa-Silva (Universidade Positivo - Campus Londrina)

2025 - semestre 1

## Na última aula...

- Objetos estáticos;
- Classes estáticas;

# Na aula de hoje...

Construtor

Sobrecarga de Construtor

Operadores de Visibilidade

Get e Set

Atividade

Atividades

Herança

Reescrita de métodos

Atividades 2

Leitura recomendada



Construtor

# Construtor de classe

E se houver a necessidade de inicializarmos um objeto com um valores, sem utilizar um método para alterar os valores de atributos?

# Construtor de classe

Construtor é o método que instancia um objeto, quando utilizamos `new String()`, estamos utilizando o construtor da classe `String`.

- Construtores possuem o mesmo nome da classe;
- É indicado que toda classe possua ao menos um construtor;
- O construtor não precisa necessariamente inicializar todos os atributos da classe.
- Uma classe pode ter vários Construtores.

# Construtor de classe

Declare um CONSTRUTOR para a classe:

```
public class Carro {  
    private String tipo;  
    private String cor;  
    private String placa;  
    private int portas;  
  
    public Carro(String tipo, String cor, String placa, int portas){  
        this.tipo = tipo;  
        this.cor = cor;  
        this.placa = placa;  
        this.portas = portas;  
    }  
}
```

Utilizando o CONSTRUTOR da classe Carro:

```
public class App {  
    public static void main(String[] args) {  
        Carro car = new Carro("Gol", "preto", "AAA3E34", 2);  
    }  
}
```

# Sobrecarga de Constructor



# Construtor de classe

Vejamos como criar vários Construtores em uma classe

```
public class Carro {  
    private String tipo;  
    private String cor;  
    private String placa;  
    private int portas;  
  
    public Carro(){}  
  
    public Carro(String tipo, String cor, String placa, int portas){  
        this.tipo = tipo;  
        this.cor = cor;  
        this.placa = placa;  
        this.portas = portas;  
    }  
}
```

# Construtor de classe

Vejamos como utilizar vários Construtores em uma classe

```
public class App {  
    public static void main(String[] args) {  
        // Utilizando construtor sem parâmetros  
        Carro car2 = new Carro();  
        // Utilizando construtor com parâmetros  
        Carro car = new Carro("Gol", "preto", "AAA3E34", 2);  
    }  
}
```

# Operadores de Visibilidade

# Operadores de visibilidade

## Público - public

Nível de acesso livre. Indica que o método ou atributo da classe é público, ou seja, pode-se acessar este atributo, ou executar esse método, por qualquer código de programa. Usamos o sinal (+) na notação UML para simbolizar essa visibilidade.

## Privado - private

Nível mais protegido. Membros declarados como private, só podem ser acessados dentro da classe em que foram declarados. Usamos o sinal (-) na notação UML para simbolizar essa visibilidade.

## Operadores de visibilidade [2]

### Protegido - protected

Nível de acesso intermediário. Serve para que o método ou o atributo seja público dentro do código da própria classe e de qualquer classe que herde daquela onde está o método ou propriedade protected. É privado e não acessível de qualquer outra parte. Usamos o sinal (#) na notação UML para simbolizar essa visibilidade.



Get e Set

# Manipulando atributos

Os atributos de modo geral devem ser protegidos, de modo que sejam manipulados apenas pela classe a qual pertencem.

- Para tanto utiliza-se o operador **private**.
- Desse modo na Classe que contém o método main, só é possível ver ou editar os atributos de objetos da classe Carro através de métodos.
- Para cada atributo com o operador de visibilidade **private**, implementamos dois métodos: get e set.
- Método **get**: Utilizado para visualizar o valor (estado) do atributo. por isso seu retorno é uma variável da mesma classe/ tipo do atributo.
- Método **set**: Utilizado para editar o valor (estado) do atributo, por isso recebe como parâmetro uma variável da mesma classe/ tipo do atributo.

# Métodos get e set da classe Carro

```
public class Carro {  
    private String tipo;  
    private String cor;  
    private String placa;  
    private int portas;  
  
    public String getTipo(){  
        return this.tipo;  
    }  
    public void setTipo(String tipo){  
        this.tipo = tipo;  
    }  
  
    public String getCor(){  
        return this.cor;  
    }  
    public void setCor(String cor){  
        this.cor = cor;  
    }  
  
    public String getPlaca(){  
        return this.placa;  
    }  
    public void setPlaca(String placa){  
        this.placa = placa;  
    }  
  
    public int getPortas(){  
        return this.portas;  
    }  
    public void setPortas(int portas){  
        this.portas = portas;  
    }  
}
```





Atividade

# Atividades

1. Crie a classe Pessoa (defina os atributos que achar necessário). Para essa classe, os atributos devem ser privados. Criem também os métodos de acesso aos atributos e também um método que imprime os dados da pessoa. Crie também uma outra classe com um método principal, que instancia um objeto da classe Pessoa, atribui valores aos atributos e acessa os seus métodos.

# Atividades

2. Crie uma classe para representar uma pseudo calculadora, que possui dois números inteiros como atributos. Nessa classe, os dois atributos devem ser privados. Por isso, criem também os métodos que irão acessar os atributos (getters e setters). Crie também métodos para realizar as seguintes operações com os dois atributos: soma, subtração, divisão e multiplicação. Crie também uma outra classe, que possui um método principal e que instancie um objeto da classe criada, e que realiza as operações disponibilizadas pelo objeto.

# Atividades

3. Escreva em Java uma classe Contador, que encapsule um valor usado para contagem de itens ou eventos (ou seja, possui um atributo do tipo inteiro que será responsável pela contagem). A classe deve possuir os métodos de acesso (get e set) e os seguintes métodos:
- Zerar(): atribui zero ao atributo da classe;
  - Incrementar(): adiciona 1 ao atributo da classe;
  - Decrementar(): subtrai um do atributo da classe;
  - Somar (valor): adiciona o valor passado por parâmetro ao atributo da classe;
  - Subtrair (valor): subtrai o valor passado por parâmetro do atributo da classe.



Atividades

# Atividades

4. Crie uma classe Retangulo em Java, que possui os seguintes atributos privados:

Lado1

Lado2

A classe deve possuir os seguintes métodos:

Construtor sem parâmetros;

Construtor que inicializa os atributos lado1 e lado 2;

Métodos de acesso aos atributos (getters e setters)

calculaArea()

calculaPerimetro()

# Atividades

5. Escreva uma classe Java para representar as contas dos clientes de um banco. Essa classe deve conter os seguintes atributos:
  1. Tipo da conta (corrente, poupança ou salário)
  2. Número da conta
  3. Nome do Cliente
  4. Saldo

Essa conta deve conter os seguintes métodos (além de 2 construtores e os métodos de acesso aos atributos):

- Sacar (valor): verifica se a conta tem saldo e subtrai o valor a ser sacado do saldo atual;
- Depositar (valor): adiciona o valor passado por parâmetro ao saldo;
- Imprimir(): imprime todos os dados da conta.

# Atividades

6. Crie uma classe Funcionário com atributos nome, sobrenome, horas trabalhadas e valor hora. A classe deve ter os seguintes métodos:
- Dois construtores (sem e com parâmetros);
  - Métodos de acesso aos atributos (getter e setters);
  - **nomeCompleto()**: retorna uma string com os atributo nome e sobrenome concatenados;
  - **calcularSalario()**: calcula e retorna o salário do funcionário no mês, multiplicando o atributo horasTrabalhadas pelo atributo valorPorHora;
  - **incrementarHoras(valor)**: adiciona um valor passado por parâmetro ao valor já existente no atributo valorPorHora;
  - **decrementaHoras (valor)**: subtrai um valor passado por parâmetro ao valor já existente no atributo valorPorHora;
  - **aumentaValorHora(porcentagem)**: reajusta o valor da hora de acordo com a porcentagem passada por parâmetro e retorna o valor reajustado.



# Atividades

7. Escreva uma classe em Java que representa um voo que acontece em determinada data e em determinado horário. Cada voo possui no máximo 100 passageiros, e a classe permite controlar a ocupação das vagas. A classe deve ter os seguintes métodos:
- Dois construtores (com e sem parâmetros);
  - Métodos de acesso aos atributos (getters e setters);
  - `proximoLivre()`: retorna o número da próxima poltrona livre. O método deve retornar -1 caso o avião esteja lotado;
  - `verificaPoltrona(numero)`: que recebe o número de uma cadeira como parâmetro e informa se a cadeira está livre ou ocupada;
  - `reservaPoltrona()`: que reserva a próxima cadeira livre e retorna o número da cadeira ocupada. O método deve retornar -1 caso o avião esteja lotado;
  - `quantidadePoltronasVagas()`: retorna a quantidade de poltronas disponíveis no voo.
  - `quantidadePoltronasOcupadas()`: retorna a quantidade de poltronas ocupadas no voo.

## Atividade - DESAFIO

8. Escreva uma classe que represente um país, que deve conter os seguintes parâmetros: código (ex.: BRA), nome, população e a sua dimensão em Km<sup>2</sup>. Adicionalmente, cada país possui uma lista de outros países com os quais ele faz fronteira. A classe deve ter os seguintes métodos:
- Dois construtores: um sem e outro com parâmetros
  - Métodos de acesso (getter/setter) para os atributos;
  - Um método que permita verificar se dois objetos representam o mesmo país. Dois países são iguais se tiverem o mesmo código;
  - Um método que informe se outro país faz fronteira com o país modelado;
  - Um método que receba um país como parâmetro e imprime o nome dos países vizinhos comuns aos países modelado.

Considere que um país tem no máximo 5 outros países com os quais ele faz fronteira.

# Funcionario.java

Vejamos a classe que registra os funcionários do banco.

```
1     private String nome;  
2     private String cpf;  
3     private double salario;  
4  
5     public double getBonificacao() {  
6         return this.salario * 0.10;  
    }
```

Crie os métodos get e set para cada atributo da classe.

# Gerente.java

Vejamos a classe que registra os Gerentes do banco.

```
1     private String cpf;
2     private double salario;
3     private int senha;
4
5     //Getters e Setters
6     public String getNome(){
7         return this.nome;
8     }
9     public void setNome(String nome){
10        this.nome = nome;
11    }
12
13    public boolean autentica(int senha) {
14        if (this.senha == senha) {
15            System.out.println("Acesso Permitido!");
16            return true;
17        } else {
18            System.out.println("Acesso Negado!");
```

Repetimos alguns trechos de código?

## Pensando...

- E se tivéssemos mais algum tipo de funcionário?
- Se for necessário guardar mais alguma informação sobre todos os funcionários? Teríamos que editar todas as classes (Funcionario, Gerente, Tipo3...)
- Existe uma maneira 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*.

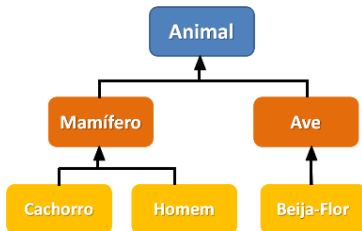


Herança

# Herança

## Definição

Quando dizemos que uma classe A é um tipo de classe B, dizemos que a classe A herda as características da classe B e que a classe B é mãe da classe A, estabelecendo então uma relação de herança entre elas.



# Herança

- Em Java, podemos usar herança simples entre classes com o `extends`.
- A nomenclatura usada é de classe mãe (*parent class*) e classe filha (*child class*), ou **superclasse** e **subclasse**.

## A herança pode ser definida como...

Herança entre classes permite que um código seja reaproveitado, de maneira que a classe filha reutilize o código da parte mãe.

A classe filha especializa a classe mais genérica, e ela é uma especialização de uma superclasse.

Herança em Java pode ser entre classes, reaproveitando membros, ou herança de uma interface, com a qual reaproveitamos interfaces de métodos. O exemplo a seguir ilustra uma hierarquia de tipos que utiliza herança em três classes.

[SILVEIRA, 2015].



# Implementando a super classe Animal

J Animal.java > ...

```
public class Animal{
    private String nome;
    private float peso;
    private int recinto;
    // Construtores
    public Animal(String nome, int recinto){
        this.nome = nome;
        this.recinto = recinto;
    }
    public Animal(){}
    // Getters and Setters
    @Override
    public String toString() {
        return "Animal [nome=" + nome + ", peso=" + peso +
            ", recinto=" + recinto + "];"
    }
}
```

# Implementando a sub classe Ave

## Detalhes da sub classe

- Note que a palavra **extends** é utilizada para informar quem é a super classe de Animal, isso dá acesso a atributos e métodos de Animal.java;
- Note que a palavra **super** é utilizada para enviar os parâmetros do construtor da super classe;
- A palavra **super** também aparece na impressão do estado de Ave.java.

# Implementando a sub classe Ave

```
public class Ave extends Animal{
    private String local;
    private boolean migracao;

    public Ave(String nome, int recinto) {
        super(nome, recinto);
    }
    public Ave() {
    }
    public String getLocal() {
        return local;
    }
    public void setLocal(String local) {
        this.local = local;
    }
    public boolean isMigracao() {
        return migracao;
    }
    public void setMigracao(boolean migracao) {
        this.migracao = migracao;
    }
    @Override
    public String toString() {
        return "Ave [local=" + local + ", migracao=" + migracao +
            ", toString()=" + super.toString() + "];"
    }
}
```



Reescrita de métodos

# Funcionario.java

Como controlar a bonificação dos funcionários?  
Vamos criar o método bonificação:

```
1      }  
2  
3      public String getNome() {
```

Porém queremos que Gerentes tenham uma bonificação de 15% e os outros funcionários uma bonificação de 10%.

**Teste a bonificação de Gerente e Funcionário.**



## Atividades 2

## Atividades 2

1. Escreva uma classe chamada Pessoa com os atributos: nome (tipo String), sexo (tipo char), idade (tipo int). Escreva agora outra classe chamada Amigo, que é uma pessoa (estende da classe Pessoa) de quem sabemos o dia de seu aniversário, atributo diaDoAniversario (tipo String). Use encapsulamento e forneça construtor padrão e construtor usando todos os atributos para as duas classes.
2. Crie uma hierarquia de classes de domínio para uma loja que venda livros, CDs e DVDs. Sobrescreva o método toString() para que imprima:
  - Para livros: nome, preço e autor;
  - Para CDs: nome, preço e número de faixas;
  - Para DVDs: nome, preço e duração.

Evite ao máximo repetição de código utilizando a palavra super no construtor e no método sobrescrito. Em seguida, crie uma classe Loja com o método main() que adicione 5 produtos diferentes (a sua escolha) a um vetor e, por fim, imprima o conteúdo do vetor.

## Atividades 2

3. Modifique o código do programa anterior, da seguinte forma:
  - Adicione um atributo que represente o código de barras do produto (é um valor obrigatório e, portanto, deve ser pedido no construtor);
  - Sobrescreva o método `equals()` retornando `true` se dois produtos possuem o mesmo código de barras;
  - Na classe `Loja`, implemente um simples procedimento de busca que, dado um produto e um vetor de produtos, indique em que posição do vetor se encontra o produto especificado ou imprima que o mesmo não foi encontrado;
  - No método `Loja.main()`, após a impressão do vetor (feita na questão anterior), escolha um dos 5 produtos e crie duas novas instâncias idênticas a ele: uma com o mesmo código de barras e outra com o código diferente. Efetue a busca deste produto no vetor utilizando as duas instâncias e verifique o resultado.



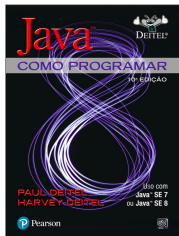
Perguntas?



Leitura recomendada

# Leitura complementar

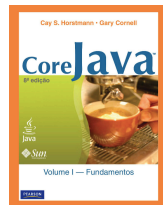
Para mais informações sobre JAVA, leia:



Java: Como programar.  
[Deitel, 2016]






Java SE 8 Programmer I  
[SILVEIRA, 2015]



Core Java.  
[Horstmann and Cornell, 20

DISPONÍVEIS NA BIBLIOTECA VIRTUAL

# Referências

-  Deitel, Paul J.; Deitel, H. M. (2016).  
*Java 8: Como programar. 10ª Edição.*  
Pearson.
-  Horstmann, C. S. and Cornell, G. (2013).  
*Core Java 2: Volume I, Fundamentals.*  
Pearson Education.
-  SILVEIRA, Guilherme; AMARAL, M. (2015).  
*Java SE 8 Programmer I: o guia para sua certificação Oracle Certified Associate.*  
Casa do Código, São Paulo, SP.