

Disciplina: LPOO

Prof. Anderson V. de Araujo

# Aula 04: Introdução aos Conceitos de Orientação a Objetos (OO)

[andvicoso@facom.ufms.br](mailto:andvicoso@facom.ufms.br)

<http://prof.facom.ufms.br/~andvicoso/>

# Programação Orientada a Objetos (POO)

- Criada por Alan Kay em 1969, através da linguagem Smalltalk
- Paradigma de programação que utiliza o conceito da criação de objetos e do relacionamento entre eles
- Provê características para o desenvolvimento rápido de software
- É o paradigma de programação mais utilizado atualmente

# Introdução à POO

- O mundo a nossa volta não está cheio de **objetos**?
  - Carros, construções, sapatos, pessoas, celulares...
- Cada um destes objetos tem a capacidade de **executar ações**
  - Acelerar, correr, ligar, enviar mensagem
  - Uma ação pode afetar outros objetos

# Classe

- Estrutura que **abstrai** um conjunto de **objetos** com **características similares**
- Define o **comportamento** de seus objetos através de **métodos** e **dados** destes objetos através de **atributos**
- Auxilia no **Reuso**
  - Ao desenhar uma planta de uma casa, um arquiteto pode usar a mesma planta para construir várias casas iguais
  - Ou como um formulário impresso e distribuído para várias pessoas preencherem, o formulário é um só (molde) mas as cópias preenchidas são diferentes (objetos)
  - Reusar classes existentes, poupa tempo e esforço

# Classes

- Encapsulam propriedades (atributos) e comportamentos (métodos)
  - Logo, atributos e métodos são **membros internos** de uma classe
- Ao escrever código Java, você está escrevendo **classes** ou **interfaces**\*


\*serão vistas mais tarde...

# Membros Internos

- **Atributos** (ou Propriedades)
  - São variáveis declaradas dentro da classe, mas fora de qualquer método
- **Métodos** (ou comportamentos)
  - São métodos internos da classe que correspondem a comportamentos associados à classe

# Responsabilidades de uma Classe

- Ao construir uma classe, temos que definir quais são as suas responsabilidades
  - **O que ela faz? Quais são suas características?**
  - **Com quais classes ela se relaciona (colabora)?**
  - Ela não pode fazer algo fora das suas responsabilidades
- Exemplo:
  - Classe ContaCorrente
    - Apresentar o saldo
    - Possibilitar um saque
    - Possibilitar um depósito

- 
- Criar nova conta
  - Armazenar todas as contas do banco
  - Gerar relatório de valor total armazenado no banco

# Objetos

- São criados a partir de classes
  - São **instâncias** de uma classe
  - Vários objetos podem ser instanciados a partir de uma classe
- A classe é o *molde* e o objeto é algo que foi *moldado* pela classe
  - A classe é similar a planta da casa e a casa (fisicamente) é o objeto criado
  - Formulário é o molde e as cópias preenchidas são os objetos
- Pode representar um objeto do mundo real ou uma abstração



# Exemplo 1

- Classe *Smartphone*
  - Atributos (propriedades):
    - Número de processadores, memória, tamanho da tela, marca...
  - Comportamentos (métodos):
    - Ligar, mandar mensagem, conectar na internet, ...
- Objetos (instâncias):
  - Iphone 6, Samsung Galaxy 5, Sony Xperia Z3, LG Nexus 5, ...

# Exemplo 1 - Java

```
package exemplo;
```

```
class Smartphone{
```

```
    int numeroDeProcs;
```

```
    double memoriaRam;
```

```
    double tamanhoTela;
```

```
    String marca;
```

Atributos  
(Propriedades)

```
    void mandarMsg(String numero, String msg){  
        //...  
    }
```

```
    void ligar(String numero){  
        //...  
    }
```

```
}
```

Métodos (Comportamentos)

## ■ Objeto Iphone 6

- numeroDeProcs = 2
- memoriaRam = 1
- tamanhoTela = 4.7
- marca = Apple

## ■ Objeto Nexus 5

- numeroDeProcs = 4
- memoriaRam = 2
- tamanhoTela = 4.95
- marca = LG

# Exercício Prático

1. Criar a classe TimeDeFutebol com os métodos e atributos
  - Atributos?
  - Comportamentos (métodos)?
  - Objetos (instâncias)?
    - Criar objetos correspondentes a classe TimeDeFutebol com as suas características definidas
2. Criar a classe aluno com as informações que você acha que um aluno deve possuir

# Ciclo de Vida de um Objeto

- Um objeto é criado usando a palavra reservada **new** (**instanciação**)
  - `Pessoa pes = new Pessoa( );`
    - A JVM busca o arquivo `Pessoa.class` no disco e o carrega para a memória
    - A JVM cria um espaço na memória do tamanho do objeto e aponta uma referência (chamada de *pes*) para o objeto criado
- Não é possível destruir objetos diretamente da memória em Java
  - Eles são removidos automaticamente da memória (pelo *Garbage Collector*) usando algumas dicas
    - Como por exemplo, quando nenhuma referência apontar para o objeto

# Resumindo...

```
Pessoa pes = new Pessoa( );
```

1 Referência chamada  
'pes'

2 Instanciação de um  
objeto Pessoa  
(chamada ao **construtor**  
padrão da classe Pessoa)

3 Atribui à variável de referência a posição  
na memória do objeto recém criado

```
class Pessoa {  
    String nome;  
    int idade;  
}
```

Memória

Pessoa

nome: \_\_\_\_  
idade: \_\_\_\_

pes



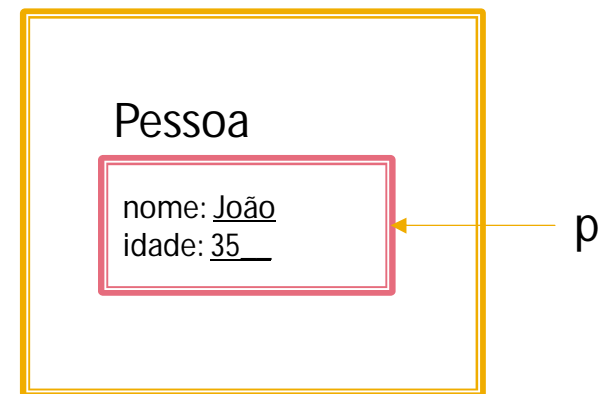
# Operador . (Ponto)

- Possibilita o acesso a membros internos de uma classe
  - Mas nem sempre isso é possível, depende do **modificador de acesso\*** de cada membro
- Exemplo:

```
Pessoa p = new Pessoa();  
p.nome = "João";  
p.idade = 35;  
p.falar();
```

\*vamos ver mais tarde...

Memória



# Método Construtor

- Toda vez que a palavra reservada **new** for usada, um método construtor vai ser chamado
  - Ele serve para “construir” um objeto
  - **Pode ter parâmetros** que indicam como esse objeto deve ser inicializado
  - **Não inicializa os atributos automaticamente**
- Se uma classe **não possuir** um construtor, o Java cria um construtor padrão (sem parâmetros) para a classe automaticamente

# Método Construtor - Exemplo

```
class Pessoa {  
    String nome;  
    int idade;  
  
    Pessoa() {  
    }  
  
    Pessoa(String novoNome, int novaIdade) {  
        nome = novoNome;  
        idade = novaIdade;  
    }  
}
```



# Variáveis

-Ué, mas a gente já não tinha aprendido tudo sobre isso?

-Nem tudo...

# Declaração de Variáveis

- Variáveis servem para armazenar valores
- Existem dois tipos de variáveis:
  - Tipos Primitivos:
    - *char, boolean, byte, short, int, long, double* ou *float*
  - **Variáveis de referência:**
    - Usadas para se referir e acessar um **objeto**

# Variáveis

- Tanto **tipos primitivos** quanto **variáveis de referência** podem ser declaradas como:
  - Atributos:
    - São variáveis declaradas dentro da classe, mas fora de qualquer método. Possuem valores padrão de inicialização
    - Classe: quando possui a palavra reservada **static**. Um **valor único** para todos os objetos/instâncias
    - Instância/Objeto: quando **não** possui a palavra reservada **static**. **Um valor para cada objeto**
  - Variáveis locais:
    - Variáveis declaradas dentro de um método/bloco de código
    - É **obrigatório inicializá-las** antes de usar
  - Parâmetros:
    - São variáveis declaradas dentro da assinatura de um método
    - Tipos primitivos: **Cópia do valor** é passada para o método
    - Variável de referência: Uma **cópia da referência** é passada para o método e não uma cópia do objeto. Ou seja, se o objeto for alterado dentro do método, ele também será alterado no método chamador

# Variáveis Locais

- São aquelas declaradas e inicializadas dentro de um método/escopo
  - E destruídas dentro do método/escopo
  - Com isso, **não podem ser acessadas** em um código **fora de onde foram declaradas**
- Embora, o valor da variável local possa ser passado para outro método, armazenando seu valor em uma variável de instância, a variável propriamente dita só vive dentro do escopo do método

# Variáveis Locais

- Devem sempre ser inicializadas antes de serem usadas

```
class TimeTravel{  
    public static void main(String[] args) {  
        int year = 2050;  
        System.out.println("The year is"+year);  
    }  
}
```

# Variável Local - Exemplos

```
class Teste {  
    int valor;  
  
    void foo() {  
        int contador = 10;  
        foo2(contador);  
    }  
    void foo2(int c) {  
        valor = c;  
    }  
    void doo(int i) {  
        contador = i;  
  
        /* não vai funcionar, pois não é possível acessar  
        contador de fora do método foo() */  
    }  
}
```

# Variáveis de Instância (Atributos)

- São definidas dentro da classe, mas fora de qualquer método
  - Não têm posição específica, mas **em geral** ficam no começo da classe
- São inicializadas quando a classe é instanciada
- São campos que pertencem a cada objeto único
- Exemplo a seguir define as variáveis de instância **nome**, **titulo** e **gerente** para o objeto **Funcionario**

```
class Funcionario {  
    String nome;  
    String titulo;  
    String gerente;  
}
```

# Variáveis de Instância de Referência a Objetos

- Quando comparadas com as variáveis primitivas não inicializadas, as **variáveis de referência** são diferentes

```
class Livro{
    String titulo;//variável de referência de instância

    String obterTitulo () {
        return titulo;
    }
    public static void main(String[] args) {
        Livro liv = new Livro();
        System.out.println("Título:" + liv.obterTitulo());
    }
}
```



# Valores Padrão para Variáveis de Instância

Tipo	Valor
boolean	false
byte	0
short	0
int	0
long	0l
char	\u0000
float	0f
double	0d
Object	null

# Variáveis de Classe

- São definidas dentro da classe, mas fora de qualquer método
- Devem ser precedidas pelo modificador ***static***
- **Pertencem a classe** onde foram declaradas, consequentemente a **todos os objetos** instanciados a partir daquela classe
  - Podem ser utilizadas sem ter que instanciar a classe a que pertencem

# Variáveis de Classe - Exemplo

```
class StaticDemo {
    static int count = 0;

    void increment() {
        count++;
    }

    public static void main(String args[]) {
        StaticDemo obj1 = new StaticDemo();
        StaticDemo obj2 = new StaticDemo();
        obj1.increment();
        obj2.increment();
        System.out.println("Obj1: count is " + obj1.count);
        System.out.println("Obj2: count is " + obj2.count);
        System.out.println("Obj: count is " + StaticDemo.count);
    }
}
```

# Métodos de Classe

- São métodos que estão associados **diretamente com a classe** e não com cada instância
  - Podem ser utilizados sem ter que instanciar a classe a que pertencem
- Devem ser precedidos pelo modificador ***static***
- E o método ***main***??
- **IMPORTANTE:** Métodos e atributos da classe (*static*) **não podem acessar** métodos e atributos da instância

# Métodos de Classe - Exemplo

```
class StaticDemo2 {  
    int x = 10;  
    static int count = 0;  
  
    void increment() {  
        count++;  
    }  
    static void decrement() {  
        count--;  
        x=20;//Erro: tentando acessar uma variável de instância  
    }  
    public static void main(String args[]) {  
        StaticDemo2 obj1 = new StaticDemo2();  
  
        obj1.increment();  
        StaticDemo2.increment();//Erro: tentando acessar um método da  
instância  
  
        obj1.decrement();  
        StaticDemo2.decrement();  
    }  
}
```

# Constantes

- São definidas através do uso de duas palavras reservadas da linguagem Java: ***static*** e ***final***
  - ***static***: indica que a variável é da classe, ou seja, só existe um valor para todos os objetos criados a partir daquela classe
  - ***final***: indica que o valor de uma variável não pode ser alterada
    - O valor das variáveis ***final*** não pode ser modificados, nem através de operadores acréscimo e decréscimo (++, --)
- Exemplo:
  - `static final double PI = 3.141592653589793;`

# Escopo de uma Variável

- Por quanto tempo uma variável existe?
  1. As variáveis **static** tem vida mais longa. São criadas quando a classe é carregada
  2. As **variáveis de instância** são criadas quando uma nova instância é criada
  3. As **variáveis locais** existem durante o tempo em que o seu método permanecer na pilha de chamadas
  4. As **variáveis de bloco** só existem durante o tempo em que o bloco de código está sendo executado

# Exemplo 1

```
class Bicycle {  
    static final MAX_SPEED = 100;  
    int gear;  
    int speed;  
  
    Bicycle(int startSpeed, int startGear) {  
        gear = startGear;  
        speed = startSpeed;  
    }  
  
    void stop(){  
        int diff = MAX_SPEED - speed;  
        //...  
    }  
}
```



# Pacotes (*Packages*)

- Em geral, as classes são agrupadas em pacotes
  - Similar a um diretório
  - Um ***package*** contém uma coleção de classes que se relacionam logicamente
- Para colocar uma classe dentro de um pacote, você deve criar o diretório, colocar a classe dentro do diretório e definir na classe o nome do diretório (pacote) logo no início da classe
- Exemplo:
  - `package diretorio;`

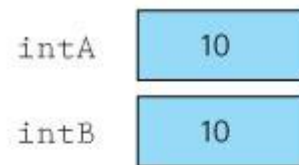
# this

- É uma palavra reservada da linguagem
- Serve para indicar a instância corrente ao **manipular membros internos** dentro da classe
- Pode ser usados para fazer chamadas entre **construtores**

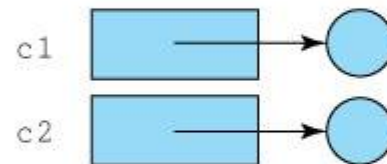
```
class Bicycle {  
    int gear;  
    int speed;  
  
    Bicycle(int sSpeed, int sGear) {  
        gear = sGear;  
        speed = sSpeed;  
    }  
  
    Bicycle(int sSpeed) {  
        this(sSpeed, 0);  
    }  
  
    void setSpeed(int speed) {  
        this.speed = speed;  
    }  
}
```

```
Bicycle bike = new Bicycle(15, 3);  
bike.setSpeed(35);
```

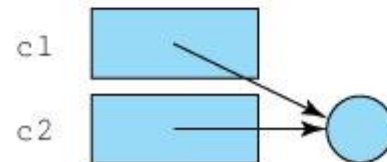
# Como Comparar Elementos?



`"intA == intB"` evaluates to true



`"c1 == c2"` evaluates to false



`"c1 == c2"` evaluates to true

# Comparando

## OPERADOR ==

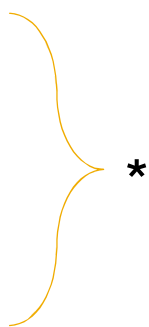
- Compara os valores em variáveis de tipos primitivos
- Em objetos, serve para comparar referências, inclusive com *null*
  - Retorna *true* se as variáveis de referências apontam para o mesmo objeto

## MÉTODO `equals`

- É um método da superclasse *Object*
- Deve-se sobrescrever o método *equals* para proceder de acordo com a regra de igualdade
  - Por exemplo, na classe *String*, o *equals* é sobrescrito para comparar caractere por caractere, verificando se ambas as strings têm todos caracteres iguais
- Se não for definido pela classe, funciona exatamente igual o operador ==

# Exemplo equals ()

```
class Pessoa {  
    public boolean equals(Object obj) {  
        // trata regra de igualdade  
        Pessoa pes = (Pessoa) obj;  
        if (pes.cpf.equals(this.cpf))  
            return true;  
        else  
            return false;  
    }  
}
```



\*Ou somente: `return pes.cpf.equals(this.cpf);`