

## II.3 DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS I

Prof. MSc. André Furlan - [ensismoebius@gmail.com](mailto:ensismoebius@gmail.com)

Etec Philadelpho Gouvêa Neto - Centro Paula Souza

2024

## Apresentação

1 - Projetar aplicativos móveis, selecionando linguagens de programação e ambientes de desenvolvimento.

**1.1** Codificar aplicativos para dispositivos móveis.

**1.2** Utilizar ambientes de desenvolvimento de software para dispositivos móveis.

**1.3** Construir interface gráfica para dispositivos móveis.

**1.4** Utilizar recursos de aparelhos celulares e tablets.

## **Desenvolvimento de aplicativos para dispositivos móveis**

- ▶ Arquiteturas e plataformas de mercado;
- ▶ Modelos de desenvolvimento:
  - ▶ Nativo;
  - ▶ Nativo multiplataforma;
  - ▶ Híbrido.
- ▶ Lojas de aplicativos.

## **Conceitos do Modelo e Plataforma de Desenvolvimento**

- ▶ Filosofia e arquitetura;
- ▶ Fundamentos da plataforma;
- ▶ Ciclo de vida e processo de desenvolvimento;
- ▶ Ferramentas (SDK, IDE/CLI, emuladores entre outros);
- ▶ Configuração do aplicativo e permissões.

## **Interface com o Usuário**

- ▶ Layouts e estilização;
- ▶ Componentes (texto, botões, imagens, listas, componentes para entrada de dados);
- ▶ Splash, diálogos e notificações;
- ▶ Navegação e roteamento.

## **Armazenamento de Dados no Lado Cliente**

- ▶ Gerenciamento de estado dos componentes;
- ▶ Armazenamento de dados offline.

- ▶ Classes e Objetos
- ▶ Encapsulamento
- ▶ Herança
- ▶ Polimorfismo

## Classes e objetos

Uma **classe** é um modelo para criar objetos que define atributos e métodos. Uma classe também pode ser definida como uma especificação de alguma coisa.

Um **atributo** é uma **característica** cujo valor pode variar de objeto para objeto.

Um **método** é uma **ação** que **pode ou não** variar de objeto para objeto dependendo se tais métodos levam ou não em consideração os valores dos atributos.

Já o **objeto** é aquilo que existe **real ou virtualmente** segundo a especificação da classe.

## **Exemplos:**

O projeto de uma casa pode ser considerado uma classe já que especifica todas as coisas que a casa fará e todas as propriedades (materiais, isolamento, pisos, etc.) que a casa terá e que, caso desejem, as pessoas que morarão nessa casa poderão ou não modificar.

Já o objeto é casa depois de construída de acordo com o projeto: Ela foi construída segundo o seu projeto original, ou seja, segundo sua classe.



# Exemplo em Java: Classe e Objeto

## Arquivo *Casa.java*

```
1 package org.dedira.oo;
2
3 public class Casa {
4     public String cor;
5     public int qtdeDePisos;
6     private int anoDeConstrucao;
7
8     public void abrirPorta() {
9         System.out.println("Nheeeeeeeee .....");
10    }
11
12    public int getAnoDeConstrucao() {
13        return this.anoDeConstrucao;
14    }
15
16    public boolean setAnoDeConstrucao(int anoDeConstrucao) {
17
18        if (anoDeConstrucao < 0) {
19            return false;
20        }
21
22        this.anoDeConstrucao = anoDeConstrucao;
23        return true;
24    }
25
26    public void exibeiInformacoes() {
27        System.out.println("Ano:" + anoDeConstrucao + " Cor:" + cor + " Pisos:" + qtdeDePisos);
28    }
29 }
```

# Exemplo em Java: Classe e Objeto

## Arquivo OO.java

```
1 package org.dedira.oo;
2
3 public class OO {
4
5     public static void main(String[] args) {
6
7         Casa minhaCasa = new Casa();
8         minhaCasa.setAnoDeConstrucao(2024);
9         minhaCasa.cor = "Rosa";
10        minhaCasa.qtdeDePisos = 1;
11
12        minhaCasa.exibeInformacoes();
13    }
14 }
15
```

Crie um novo programa que represente a entidade trabalhador. Essa entidade deve ter 10 atributos e 5 ações. Pelo menos 4 das 5 ações devem levar em consideração ou usar pelo menos 5 atributos cada uma.  
O programa deve rodar devidamente.

Um programa em Java é composto por classes que são compostas por **propriedades** e **métodos**.

Uma **classe** pode ser definida como um modelo que determina como o objeto se comporta e quais suas características.

```
1 // Linhas que comecam com "//" sao comentarios
2 // Um comentario nao e executado, serve apenas
3 // como uma manual de instrucoes para facilitar
4 // o entendimento do codigo
5
6 // O corpo da classe e tudo aquilo
7 // que esta dentro das chaves
8 // A linha abaixo define a assinatura de uma classe.
9 public class OO {
10     // O corpo da classe fica aqui
11 }
12
```

# Definições básicas

Um **método** é uma ação que o objeto pode realizar. Dentro do contexto da programação um método é uma função ou procedimento, ou seja, um bloco de código que tem parâmetros e que pode retornar ou não algum valor.

```
1  public class OO {
2
3      // Um metodo similar a um procedimento. Perceba que ele retorna "void", ou seja, um valor vazio
4      public void abrirPorta() {
5          System.out.println("Nheeeeeeeee.....");
6      }
7
8      // Um metodo similar a uma funcao. Esse retorna "int", ou seja, um numero inteiro
9      public int getAnoDeConstrucao() {
10         return 10;
11     }
12
13     // Esse eh um metodo que retorna um "boolean", ou seja, um valor verdadeiro ou falso.
14     // Perceba que esse metodo eh uma funcao que recebe parametros
15     public boolean setAnoDeConstrucao(int anoDeConstrucao) {
16
17         if (anoDeConstrucao < 0) {
18             return false;
19         }
20
21         return true;
22     }
23 }
24
```

Uma **propriedade** é uma variável que é visível para todo o objeto, ou seja, todos os métodos e outras propriedades **internas** do objeto podem enxergar essa variável.

```
1  public class Casa {  
2      // Essa eh uma propriedade do tipo "String", ou seja , um texto  
3      public String cor;  
4      // Essa tambem eh uma propriedade do tipo "int", ou seja , um numero inteiro  
5      public int qtdeDePisos;  
6  }  
7
```

Os tipos primitivos em Java são aqueles que representam valores simples. Aqui estão os principais tipos primitivos:

- ▶ `byte`: representa números inteiros de 8 bits.
- ▶ `short`: representa números inteiros de 16 bits.
- ▶ `int`: representa números inteiros de 32 bits.
- ▶ `long`: representa números inteiros de 64 bits.
- ▶ `float`: representa números de ponto flutuante de precisão simples.
- ▶ `double`: representa números de ponto flutuante de dupla precisão.
- ▶ `char`: representa um único caractere Unicode.
- ▶ `boolean`: representa valores lógicos verdadeiro ou falso.



# Tipos de referência

Os tipos de referência em Java são baseados em classes e são usados para criar objetos. Podemos usar inclusive as próprias classes criadas por nós! Aqui estão alguns exemplos comuns de tipos de referência:

- ▶ `String`: representa uma sequência de caracteres.
- ▶ `Scanner`: usado para receber entrada do usuário.
- ▶ `ArrayList`: implementação de uma lista redimensionável.
- ▶ `Object`: a superclasse (veremos mais adiante o que é isso) de todos os tipos de objetos em Java.

## Encapsulamento

Antes de entrar mais profundamente no assunto de encapsulamento precisamos entender o que são as visibilidades das propriedades e métodos em orientação a objetos.

- ▶ **Visibilidade Pública:** indica que aquele atributo ou método será acessível para outros objetos um atributo público também é herdável (veremos sobre herança mais adiante)
- ▶ **Visibilidade privada:** indica que o atributo ou método não será acessível para outros objetos um atributo privado não é herdável.
- ▶ **Visibilidade Protegida:** indica que o atributo ou método protegido também não é acessível a outros objetos, no entanto, eles são herdáveis.

Em orientação a objetos, dependendo do estilo de programação da programadora ou do programador, às vezes é desejável **"esconder"** alguns atributos e métodos para assim **evitar algum acesso ou modificação acidentais**.

Isso é válido principalmente para os atributos pois alguns podem necessitar de algum tipo de **validação** antes de serem modificados. Se quisermos fazer isso é necessário definir sua visibilidade como **privada**, dessa forma, não haverá modificações indesejadas. No entanto isso deixa o atributo inacessível impossibilitando qualquer modificação que seja necessária, para resolver esse problema implementamos o que se convencionou dizer chamar como **"getters"** e **"setters"** que são métodos públicos que terão o papel de validar os dados de entrada e alterar os atributos (setters) ou ler os valores que os mesmos contém (getters).

# Encapsulamento

```
1  public class Casa {
2
3      private String cep; // Propriedade encapsulada
4      private int altura; // Propriedade encapsulada
5
6
7      // Getters
8      public String getCep() {
9          return cep;
10     }
11
12     public int getAltura() {
13         return altura;
14     }
15
16     // Setters
17     public void setCep(String cep) {
18         this.cep = cep;
19     }
20
21     public void setAltura(int altura) {
22         this.altura = altura;
23     }
24 }
25
26
```

# Exemplo em Java: Encapsulamento

```
1 public class ContaBancaria {  
2     private double saldo;  
3  
4     public double getSaldo() {  
5         return saldo;  
6     }  
7  
8     public void depositar(double valor) {  
9         saldo += valor;  
10    }  
11 }  
12
```

Faça um programa que contenha duas classes a primeira classe deve conter o método principal e ele deve instanciar a segunda classe, já a segunda classe deve conter três atributos um público e dois privados. Implemente o encapsulamento desses atributos privados.

## Herança e polimorfismo



- ▶ Permite que uma classe herde atributos e métodos de outra classe
- ▶ Promove a reutilização de código

- ▶ Permite que métodos seja sobrecarregados

# Exemplo em Java: Moradia.java

```
1 public class Moradia {
2     public String cor;
3     public int qtdeDePisos;
4     protected int anoDeConstrucao;
5     private String cnpjDoCapitalistaSafado;
6     public int getAnoDeConstrucao() {
7         return this.anoDeConstrucao;
8     }
9     public boolean setAnoDeConstrucao(int anoDeConstrucao) {
10         if (anoDeConstrucao < 0) return false;
11         this.anoDeConstrucao = anoDeConstrucao;
12         return true;
13     }
14     public void abrirPorta() {
15         System.out.println("Nheeeeeeeee.....");
16     }
17     public void exibelnformacoes() {
18         System.out.println("Ano:" + anoDeConstrucao + " Cor:" + cor + " Pisos:" + qtdeDePisos + " Cnpj: " +
19             this.cnpjDoCapitalistaSafado);
20     }
21     public String getCnpjDoCapitalistaSafado() {
22         return cnpjDoCapitalistaSafado;
23     }
24     public boolean setCnpjDoCapitalistaSafado(String cnpjDoCapitalistaSafado) {
25         if (cnpjDoCapitalistaSafado.length() == 14) {
26             this.cnpjDoCapitalistaSafado = cnpjDoCapitalistaSafado;
27             return true;
28         }
29         return false;
30     }
31 }
```

# Exemplo em Java: Herança e polimorfismo - Casa.java

```
1 public class Casa extends Moradia {
2
3     // Isso aqui eh um metodo construtor
4     public Casa() {
5
6         // Isso aqui vai da erro
7         // this.cnpjDoCapitalistaSafado
8
9         // A propriedade cnpjDoCapitalistaSafado eh privada
10        // portanto nao pode ser herdada.
11        // Mas setCnpjDoCapitalistaSafado eh publico
12        // e herdavel
13        this.setCnpjDoCapitalistaSafado("00000000000000");
14    }
15
16    // Isso aqui eh uma annotation indicando que o
17    // metodo esta sendo sobrecarregado
18    @Override
19    public void abrirPorta() {
20        System.out.println("Abre a porta mariquinha");
21    }
22
23 }
24
```

# Exemplo em Java: Herança e polimorfismo - OO.java

```
1 public class OO {
2     public static void main(String[] args) {
3
4         Moradia minhaCasa = new Moradia();
5         minhaCasa.setAnoDeConstrucao(2024);
6         minhaCasa.cor = "Rosa";
7         minhaCasa.qtdeDePisos = 1;
8         minhaCasa.abrirPorta();
9         minhaCasa.exibeInformacoes();
10
11         // Os mesmos metodos estao sendo chamados mas os resultados sao diferentes gracias a heranca e ao
12         // polimorfismo
13         Casa casaAmarela = new Casa();
14         casaAmarela.setAnoDeConstrucao(2017);
15         casaAmarela.cor = "Amarela";
16         casaAmarela.qtdeDePisos = 2;
17         casaAmarela.abrirPorta();
18         casaAmarela.exibeInformacoes();
19
20         // Os mesmos metodos estao sendo chamados mas os resultados sao diferentes gracias a heranca e ao
21         // polimorfismo e mais, tambem ha um metodo a mais: subirDeElevador()
22         Apartamento meuAp = new Apartamento();
23         meuAp.setAnoDeConstrucao(2017);
24         meuAp.cor = "Branco";
25         meuAp.qtdeDePisos = 1;
26         meuAp.abrirPorta();
27         meuAp.subirDeElevador();
28         meuAp.exibeInformacoes();
29     }
30 }
```

Implemente três classes: A primeira deve conter o método principal, no método principal se deve instanciar a segunda e a terceira classe.

A Segunda classe deve ser uma classe comum que não estende qualquer outra classe.

A Terceira classe deve estender a segunda, sobrescrever algum de seus métodos, e implementar um método próprio.

Use o exemplo informado nos slides anteriores como base.