Desenvolvimento Mobile 1
Aula 06

Prof. Me Daniel Vieira

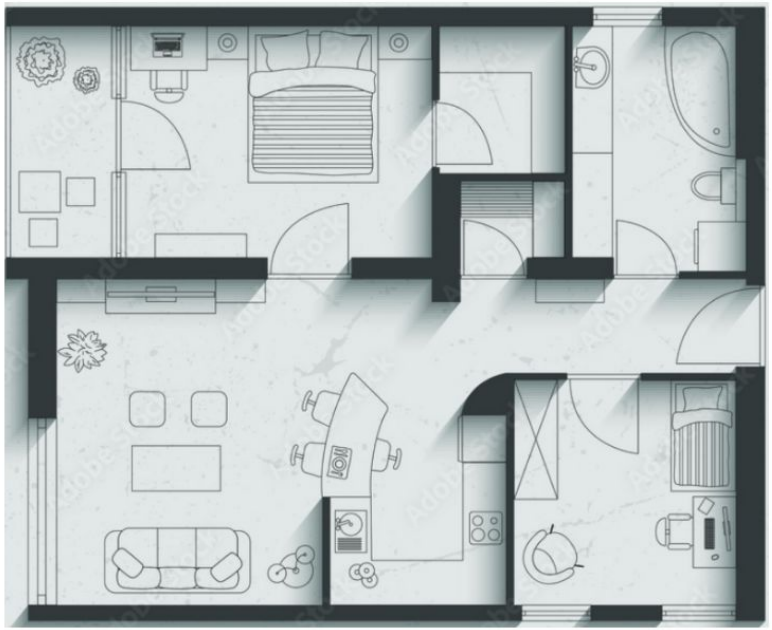
SENAI

Agenda

1 - Programação orientada a objeto
2 - Classe abstrata
3 - Poliformismo
4 Get e setter
5 - Exercícios

Programação orientada a objeto

Classe: Casa



Objeto: Casa

Polimorfismo

- Polimorfismo:

Permite que objetos de classes diferentes sejam tratados como objetos de uma mesma superclasse.

O comportamento do método pode variar dependendo do objeto.

Polimorfismo

O **polimorfismo** é um conceito fundamental da **programação orientada a objetos (POO)** que permite que um mesmo método tenha diferentes comportamentos dependendo da classe que o implementa. Em Dart, isso é alcançado principalmente através da **herança** e da **implementação de interfaces**.

📌 Como funciona?

O polimorfismo permite que uma classe filha **substitua** (override) os métodos da classe pai, garantindo que cada classe possa ter uma implementação específica do método herdado.

Polimorfismo

O **polimorfismo** é um conceito fundamental da **programação orientada a objetos (POO)** que permite que um mesmo método tenha diferentes comportamentos dependendo da classe que o implementa. Em Dart, isso é alcançado principalmente através da **herança** e da **implementação de interfaces**.

📌 Como funciona?

O polimorfismo permite que uma classe filha **substitua** (override) os métodos da classe pai, garantindo que cada classe possa ter uma implementação específica do método herdado.

Polimorfismo

```
// Classe base
class Animal {
  void fazerSom() {
    print("O animal faz um som");
  }
}
```

Polimorfismo

```
// Classe derivada: Cachorro
class Cachorro extends Animal {
  @override
  void fazerSom() {
    print("O cachorro late: Au Au!");
  }
}
```

Polimorfismo

```
// Classe derivada: Gato
class Gato extends Animal {
  @override
  void fazerSom() {
    print("O gato mia: Miau!");
  }
}
```

Polimorfismo

```
void main() {
  Animal meuAnimal = Cachorro(); // Polimorfismo em ação
  meuAnimal.fazerSom(); // Saída: O cachorro late: Au Au!

  meuAnimal = Gato();
  meuAnimal.fazerSom(); // Saída: O gato mia: Miau!
}
```

Polimorfismo

```
void main() {
  Animal meuAnimal = Cachorro(); // Polimorfismo em ação
  meuAnimal.fazerSom(); // Saída: O cachorro late: Au Au!

  meuAnimal = Gato();
  meuAnimal.fazerSom(); // Saída: O gato mia: Miau!
}
```

Polimorfismo

```
1. /*
2. Classe forma com o método desenhar
3. Classe círculo herda método desenhar com método.
4. O comportamento do método pode variar de acordo com o objeto.
5. */
6. class Forma {
7.   void desenhar() {
8.     print("Desenho genérico");
9.   }
10. }
11.
```

Polimorfismo

```
12. class Circulo extends Forma {
13.   @override
14.   void desenhar() {
15.     print("Desenhando um círculo");
16.   }
17. }
18.
19. void desenharForma(Forma forma) {
20.   forma.desenhar();
21. }
22.
```

Abstração

- Abstração:

Foca nos aspectos essenciais de um objeto, escondendo os detalhes internos.

Usa classes abstratas e interfaces para definir comportamentos comuns.

Abstração

```
 // Interface (ou contrato)
abstract class Forma {
  double calcularArea();
}

// Classe que implementa a interface
class Circulo implements Forma {
  double raio;
  Circulo(this.raio);

  @override
  double calcularArea() {
    return 3.14 * raio * raio;
  }
}
```

Abstração

```
// Outra classe que implementa a interface
class Retangulo implements Forma {
  double largura, altura;
  Retangulo(this.largura, this.altura);

  @override
  double calcularArea() {
    return largura * altura;
  }
}
```

Abstração

```
void main() {
  Forma forma1 = Circulo(5);
  Forma forma2 = Retangulo(4, 6);

  print("Área do círculo: ${forma1.calcularArea()}");
  print("Área do retângulo: ${forma2.calcularArea()}");
}
```

Abstração

```
1. //Classe abstrata animal
2. abstract class Animal {
3.   void emitirSom();
4. }
5.
6. class Gato implements Animal {
7.   @override
8.   void emitirSom() {
9.     print("Miau!");
10.   }
11. }
12.
```

Abstração

Crie uma classe abstrata denominada “Alimentar” com três métodos:
Separar ingredientes
Pegar tigela
Preparar comida
Essa classe abstrata deve ser implementada na classe “Filha”.

Abstração

O polimorfismo em Dart permite:
Criar código mais flexível e reutilizável.
Tratar objetos de diferentes tipos de forma unificada.
Garantir que classes diferentes implementem métodos de maneira específica.

Isso é útil quando se deseja trabalhar com coleções de objetos diferentes, permitindo que um mesmo método tenha comportamentos distintos dependendo do tipo real do objeto! 🚀

Abstração

```
abstract class Alimentar {
  void separarIngredientes();
  void pegarTigela();
  void prepararComida();
}
```

Abstração

```
class Filha implements Alimentar {
  @override
  void separarIngredientes() {
    print('Ingredientes separados.');
  }

  @override
  void pegarTigela() {
    print('Tigela pega.');
  }

  @override
  void prepararComida() {
    print('Comida preparada.');
  }
}
```

Get e Setter

Os getters e setters em Dart são métodos especiais usados para acessar e modificar os atributos (ou propriedades) de uma classe, fornecendo maior controle sobre como os dados são manipulados.

- Utilidade de Getters e Setters

Get e Setter

Encapsulamento: Protegem os atributos da classe, permitindo acesso e manipulação controlados.
Validação e Lógica: Permitem incluir lógica ao acessar ou alterar os valores, como verificar se o novo valor é válido antes de atribuí-lo.
Interface Limpa: Escondem a complexidade, oferecendo uma interface de acesso parecida com a de atributos normais.
Imutabilidade Controlada: Podem impedir alterações diretas em atributos que precisam ser apenas lidos (getter sem setter).
Sintaxe de Getters e Setters em Dart
Getter: Um método que retorna o valor de um atributo.
Setter: Um método que define (ou altera) o valor de um atributo.
A palavra-chave get é usada para definir um getter, e set é usada para definir um setter.

Get e Setter

1. /*Encapsulamento:
2.
3. O atributo _preco é privado (só pode ser acessado dentro da classe).
4. O getter e setter permitem acessá-lo e modificá-lo de forma controlada.
5. Validação no Setter:
6.
7. Antes de alterar o valor de _preco, o setter verifica se o valor é maior que zero. Se não for, ele rejeita a alteração.
8. Uso Prático:
9.
10. O acesso ao preço é feito de forma transparente com o getter (produto.preco), como se fosse um atributo normal.
11. A atribuição do preço utiliza o setter (produto.preco = valor), permitindo que a validação seja aplicada automaticamente.
12. */

Get e Setter

```
13. class Produto {
14.   // Atributo privado (convenção: usa-se '_' para indicar privado)
15.   double _preco = 0.0;
16.
17.   // Getter: Retorna o valor do atributo _preco
18.   double get preco => _preco;
19.
20.   // Setter: Define um novo valor para _preco com validação
21.   set preco(double novoPreco) {
22.     if (novoPreco > 0) {
23.       _preco = novoPreco;
24.     } else {
25.       print("O preço deve ser maior que zero.");
26.     }
27.   }
28. }
29.
```

Get e Setter

```
30. void main() {
31.   // Criando um objeto da classe Produto
32.   Produto produto = Produto();
33.
34.   // Usando o setter para definir o preço
35.   produto.preco = 50.0; // Valor válido
36.   print("Preço do produto: R\$${produto.preco}"); // Saída: R$50.0
37.
38.   produto.preco = -10.0; // Valor inválido
39.   // Saída: "O preço deve ser maior que zero."
40.
41.   print("Preço do produto: R\$${produto.preco}"); // Saída: R$50.0 (valor anterior não foi
alterado)
42. }
43.
```

Get e Setter

Vantagens de Usar Getters e Setters:

- Segurança: Impede que dados sejam alterados diretamente sem passar por regras ou validações.
- Flexibilidade: Permite implementar lógica adicional sem mudar a interface pública da classe.
- Manutenção: Facilita a evolução do código, pois as mudanças podem ser feitas nos getters e setters sem alterar a interface da classe.

Exercícios

1. Crie uma classe abstrata “Máquina industrial” com os seguintes métodos:
Nome – Nome da máquina
Potência da máquina
Status – booleano
Métodos abstratos
Ligar() – Um método abstrato que define como a máquina é ligada
Desligar () – Método abstrato que define como a máquina é desligada

Exercícios

2. Crie duas subclasses concretas de máquina industrial Prensa e Robô solda.
Prensa deve ter um atributo adicional chamado “Pressão em toneladas” e os métodos ligar e desligar devem exibir mensagens adequadas.
Robô solda deve ter um atributo chamado tipo de solda(String) para especificar o tipo de solda que realiza. Os métodos ligar e desligar devem exibir mensagens adequadas.

Exercícios

3. Crie a classe “Pessoa” com getters e setters para o nome e idade da pessoa. Os setters permitem configurar esses atributos com verificações de validade. A classe também possui um método para mostrar informações para exibir as informações da pessoa.

Exercícios

4 Crie uma classe abstrata chamada “Automóveis” com nome, cor, ano como atributos.
5 Crie uma classe chamada “Carro” herdando características da classe abstrata “Automóveis”.

6. Crie métodos abstratos na classe abstrata como colocar o cinto, ligar o carro, desligar o carro e dirigir.

7. Crie uma classe concreta “Carro” implementando os métodos abstratos exibindo mensagens adequadas.

Obrigado!

Prof. Me Daniel Vieira

Email: danielvieira2006@gmail.com

Linkedin: Daniel Vieira

Instagram: Prof daniel.vieira95