
SENAI

Aula 05
Orientação a Objetos

SENAI

Na aula de hoje

- Resolução dos exercícios da aula passada;
- Programação Estruturada vs. Programação Orientada a Objetos;
- Conceito de Programação Orientada a Objetos;
- Tópicos de Orientação a Objetos;
- Exemplos de Código;
- Pilares da Orientação a Objetos;
- Exercícios conceituais.

Programação Estruturada vs. POO

SENAI

Orientação a Objetos
Programação Estruturada

Conceito

A programação estruturada organiza o programa em uma série de blocos de código bem definidos, utilizando sequências, seleções (if/else) e iterações (loops) para controlar o fluxo de execução.

Este método enfatiza a clareza, a modularidade e a facilidade de leitura do código.

Principais características

Divisão do programa em módulos
Facilita a organização e reutilização de código.

Ênfase na lógica procedural
O programa descreve os passos para realizar uma tarefa.

Utilização de estruturas de controle
Sequências, seleções e iterações controlam o fluxo de execução.

Foco na clareza e legibilidade do código
Facilita a manutenção e depuração.

Orientação a Objetos
Programação Estruturada

Casos de uso
A programação estruturada ainda é amplamente utilizada em diversos cenários, como:

Scripts de automação
Automatização de tarefas repetitivas.

Processamento de dados
Manipulação e análise de dados em larga escala.

Programação de sistemas embarcados
Controle de dispositivos com recursos limitados.

Desenvolvimento de algoritmos
Implementação de soluções para problemas computacionais.

Orientação a Objetos
Programação Orientada a Objetos

SENAI

Conceito

A Programação Orientada a Objetos (POO) é um paradigma de programação que se baseia na ideia de "objetos", os quais são entidades que possuem tanto dados (atributos) quanto comportamentos (métodos) associados a eles.

Os objetos encapsulam dados (atributos) e comportamentos (métodos), representando entidades do mundo real.

Principais características

Abstração
Representação simplificada de entidades complexas.

Encapsulamento
Dados e métodos são protegidos dentro do objeto

Herança
Criação de novas classes a partir de classes existentes, promovendo reutilização de código.

Polimorfismo
Objetos de diferentes classes podem responder ao mesmo método de maneiras diferentes.

Orientação a Objetos
Programação Orientada a Objetos

Casos de Uso
A POO é amplamente utilizada em diversos tipos de projetos, como:

Desenvolvimento de aplicações web
Criação de sistemas complexos e escaláveis.

Desenvolvimento de jogos
Modelagem de personagens, objetos e interações.

Aplicações mobile
Criação de interfaces gráficas e funcionalidades avançadas.

Sistemas de gerenciamento de dados
Organização e manipulação eficiente de grandes volumes de dados.

Diferentemente do paradigma procedural, onde o código é organizado em torno de funções e procedimentos que operam em dados, na POO, o código é organizado em torno de objetos que interagem uns com os outros através de mensagens., e assim por diante.

Orientação a Objetos
Prog. Estruturada vs. POO

SENAI

Principais diferenças entre os tipos de estrutura

Característica	Programação Estruturada	Programação Orientada a Objetos
Foco	Procedimentos e funções	Objetos e suas interações
Estrutura	Blocos de código sequenciais	Classes e objetos
Reutilização de código	Limitada a funções e módulos	Alta, através de herança e polimorfismo
Modelagem de dados	Variáveis e estruturas de dados	Objetos com atributos e métodos

Orientação a Objetos
Prog. Estruturada vs. POO

Vantagens e Desvantagens

Paradigma	Vantagens	Desvantagens
Programação Estruturada	Simplicidade, facilidade de aprendizado	Menos flexível para sistemas complexos, reutilização limitada
Programação Orientada a Objetos	Modelagem intuitiva, alta reutilização de código, maior flexibilidade	Curva de aprendizado mais acentuada, maior complexidade inicial

Orientação a Objetos
Prog. Estruturada vs. POO

Orientação a Objetos
Conclusão

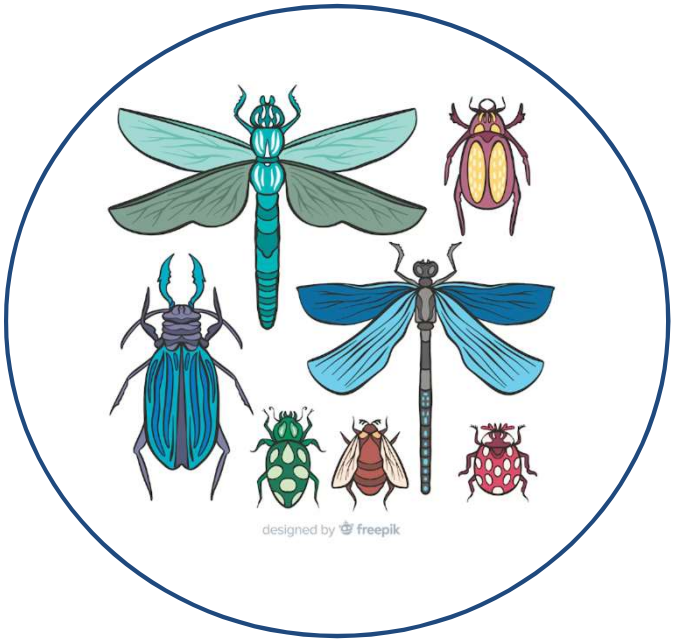
A escolha entre programação estruturada e programação orientada a objetos **depende da natureza do projeto**, da experiência da equipe de desenvolvimento e dos requisitos de desempenho e escalabilidade.

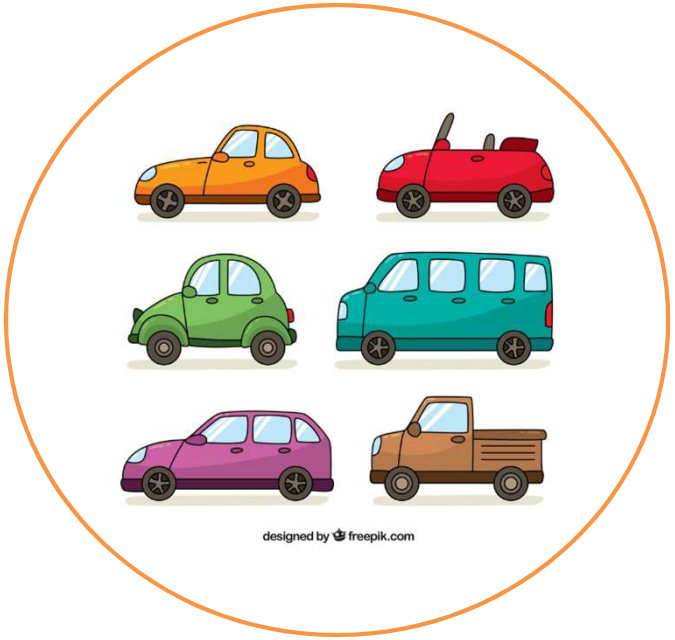
Ambos os paradigmas oferecem ferramentas poderosas para a criação de software, e a compreensão de seus conceitos e diferenças é fundamental para o desenvolvimento de soluções eficientes e robustas.

Conceito de POO

SENAI

Orientação a Objetos
Classe





Orientação a Objetos
Classe

Classe

Uma classe de objetos descreve um grupo de objetos com propriedades (atributos) similares, comportamento (operações) similares, relacionamentos comuns com outros objetos e uma semântica comum.

É um modelo ou um plano para criar objetos. Ela define os atributos (dados) e métodos (comportamentos) que os objetos desse tipo terão.

Por exemplo, se considerarmos uma classe "Carro", ela pode ter atributos como marca, modelo e ano, e métodos como acelerar, frear e ligar os faróis. Cada objeto criado a partir dessa classe "Carro" terá seus próprios valores para os atributos e poderá executar os métodos definidos na classe.

Chamamos essa criação de instanciação da classe, como se estivéssemos usando esse molde (classe) para criar um objeto.

Exemplo:

```
// Definição da classe
public class Carro
{
    // Campos (atributos) da classe
    public string Marca;
    public string Modelo;
    public int Ano;
```

O código fornecido define uma classe chamada Carro que possui três atributos: marca, modelo e ano.

Dica
Para manter um código mais organizado e seguir as convenções de nomenclatura comuns em C#, é recomendável ter cada classe em seu próprio arquivo. Isso facilita a navegação no código e melhora a legibilidade.

Orientação a Objetos
Classe

Uma classe precisa responder 3 perguntas:

Coisas que tenho ? — **Atributo**
- modelo
- fabricante
- cor
- motor
- potencia

Propriedade
Característica
Dados

Coisas que eu faço ? — **Método**
- andar
- parar
- acelerar
- virar esquerda
- virar direita

Procedimentos
Rotinas internas

Com estou agora ? — **Estado**
- ligado
- desligado
- parado
- em movimento

Característica no momento da análise do objeto

Orientação a Objetos
Atributo

Marca

Ano

Modelo

Orientação a Objetos
Atributo

Atributo

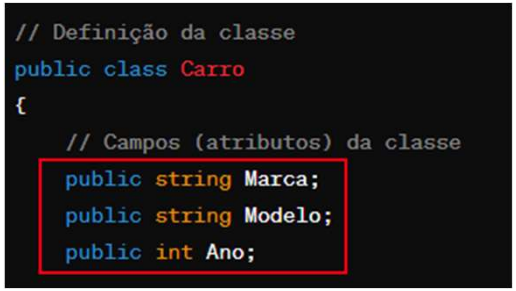
Um atributo é um valor de dado assumido pelos objetos de uma classe.

Um atributo é uma variável associada a uma classe que armazena dados relacionados a cada objeto criado a partir dessa classe.

Cada objeto criado a partir da classe possui seus próprios valores para os atributos, permitindo assim a representação de diferentes estados para cada instância da classe.

- Nome, idade e peso são exemplos de atributos de objetos Pessoa.
- Cor, peso e modelo são possíveis atributos de objetos Carro. Cada atributo tem um valor para cada instância de objeto.

Exemplo:



Neste exemplo, a classe Carro possui três atributos: marca, modelo e ano.

Cada objeto criado a partir desta classe terá seu próprio conjunto de valores para esses atributos.

Orientação a Objetos

Objeto

Orientação a Objetos
Objeto

Objeto

Um objeto é uma instância de uma classe

Em termos simples, um objeto é uma entidade que possui estado (atributos) e comportamento (métodos)

Estado: Representado atributos (variáveis) do objeto.
Comportamento: Representado pelos métodos (funções) do objeto.

Quando construímos uma casa, não começamos a casa de qualquer maneira, colocando tijolos em qualquer lugar, você faz uma planta para decidir a metragem da casa, o material utilizado, quantos quartos, quantos banheiros, etc.
A nossa classe é a planta dessa casa, que define todos os atributos, variáveis da nossa casa.
Na classe podemos ter comportamentos, métodos, como por exemplo, construir (uma função).
O objeto vai ser a nossa casa, construída com base nessa planta.

Orientação a Objetos
Objeto

Objetos

Objeto é definido neste modelo como um conceito, abstração ou coisa com limites e significados bem definidos para a aplicação em questão.

Ele representa uma entidade do mundo real ou abstrato que possui características (atributos) e comportamentos (métodos).

Em outras palavras, **um objeto é uma variável que pode armazenar dados e manipulá-los através de métodos específicos associados à sua classe.**

Por exemplo, se tivermos uma classe "Carro", um objeto dessa classe poderia representar um carro específico com suas características únicas, como cor, modelo e velocidade atual, e poderia realizar ações como acelerar, frear e ligar os faróis.

Exemplo:

```
// Método construtor
public Carro(string marca, string modelo, int ano)
{
    Marca = marca;
    Modelo = modelo;
    Ano = ano;
}
```

O código fornecido define um construtor que inicializa esses atributos ao criar um objeto da classe.

No contexto de um construtor em C#, this.marca = marca; está atribuindo o valor do parâmetro marca à variável de instância marca da classe Carro.

Orientação a Objetos
Operações

Operações

Uma operação é uma função ou transformação que pode ser aplicada a ou por objetos em uma classe.

Por exemplo, abrir, salvar e imprimir são operações que podem ser aplicadas a objetos da classe Arquivo. Todos objetos em uma classe compartilham as mesmas operações.

Operações podem envolver a manipulação dos atributos do objeto, interações com outros objetos, ou retornar valores específicos. Por exemplo, em uma classe "Conta Bancária", operações comuns podem incluir depositar, sacar e verificar o saldo.

Elas representam o comportamento dos objetos e ajudam a encapsular a lógica de negócios dentro de uma classe.

Exemplo:

```
public void Acelerar(int aumento)
{
    Velocidade += aumento;
    Console.WriteLine("O carro está acelerando. Velocidade atual: ");
    Console.WriteLine($"{Velocidade} km/h");
}
```

Método acelerar: Este é um método da classe Carro que representa uma operação de aceleração. Ele recebe um parâmetro aumento que indica o quanto a velocidade do carro deve aumentar.
Dentro do método, a velocidade atual do carro é aumentada pelo valor de aumento.
Em seguida, uma mensagem é exibida na saída padrão, mostrando que o carro está acelerando e qual é sua velocidade atual.

Orientação a Objetos
Métodos (funções)

Método ou função:

É um bloco de código que realiza uma tarefa específica, que pode ser reutilizada em diversas partes do programa.

No Java é usado o termo Método, por ser uma linguagem orientada a objeto, que é a função associada a classe e ao objeto.

Então quando você vai criar um método precisamos seguir a sintaxe de criação do método:

[modificador de acesso] [tipo de retorno] [nome do método] [parâmetros]

Exemplo:
public void contruir ()

Orientação a Objetos
Modificadores de acesso

Temos três tipo de modificadores de acesso:

Public: acessível em qualquer local do código
Private: acessível apenas na classe que foi declarado
Protected: acessível dentro do mesmo pacote ou subclasse (dentro da mesma pasta)

Orientação a Objetos
Tipos de retorno

Define o tipo de dado que o método devolverá (int, String, double, Objeto)
Quando criamos o método construir() ele só retornou a estrutura da casa, que a construção, para isso usamos a palavra **void (quando não retorna nenhum valor)**

```
public void pintar()
{
   System.out.println("A casa foi pintada de: " + cor);
}
```

Podemos usar o método para retornar um cálculo, e neste caso o retorno será (int, String, double, Objeto), o método pode retornar qual um dos tipos primitivos ou não, além de um Objeto

Na Classe PlantaCasa

```
//Método Somar Metragem
public int somarMetragem()
{
   return metragem * numeroBanheiros + numeroQuartos;
}
```

Na Classe Casa

```
//Retorno do resultado
int resultado = casa.somarMetragem();
System.out.println("A metragem da casa é " + resultado);
```

Orientação a Objetos
Parâmentros de método

Os parâmentros são variáveis que são passados para o método.
Estes parâmetros são definidos dentro dos parênteses logo após o nome do método.

Um método pode ter parâmetros ou pode não ter.

Então os parâmetros são dados que serão usados dentro do método para realizar a ação correspondente.

Orientação a Objetos
Método instância e estáticos

Método instância: precisam ser chamados a partir de uma instância (objeto) da classe.
Por exemplo:
- contruir()
- pintar()

Estes dependem da PlantaCasa

Método estático: podem ser chamados diretamente a partir da classe, sem a necessidade de criar um objeto.
Por exemplo:

```
public class soma
	{
		public static int somar(int a, int b)
			{
				return a+b;
			}

		public static String converterString(Integer a)
			{
				return a.toString();
			}

	}
```

Orientação a Objetos
Método

Método

Um método é a implementação de uma operação para uma classe.

Métodos definem o comportamento dos objetos da classe, especificando as operações que podem ser realizadas sobre eles.

Um método pode realizar diversas ações, manipular os atributos do objeto, interagir com outros objetos, ou retornar valores específicos. Por exemplo, um método de uma classe "Carro" pode ser "ligarMotor()", que realiza a ação de ligar o motor do carro.

Eles são uma parte essencial da definição de uma classe e ajudam a encapsular o comportamento associado a um conjunto de objetos relacionados.

Facilita a manutenção do código, pois as alterações internas na implementação da classe não afetam o restante do programa, desde que a interface pública permaneça a mesma.

Para implementar o encapsulamento em Java, por exemplo, os atributos de uma classe geralmente são declarados como privados e acessados por métodos públicos de leitura (métodos get) e escrita (métodos set), que controlam o acesso aos dados e aplicam regras de validação, se necessário.

```
public void Acelerar(int aumento)
{
    Velocidade += aumento;
    Console.WriteLine("O carro está acelerando. Velocidade atual: ");
    Console.WriteLine($"{Velocidade} km/h");
}
```

Orientação a Objetos
Interface

Interface

Uma interface é um tipo de estrutura que define um conjunto de métodos abstratos que uma classe deve implementar. Ela define um contrato que as classes devem cumprir ao implementar a interface.

Uma classe pode implementar múltiplas interfaces, o que é conhecido como herança múltipla de tipo. Isso permite que uma classe adote vários comportamentos diferentes, definidos pelas interfaces que ela implementa.

Em resumo, uma interface em POO define um contrato para as classes, especificando quais métodos devem ser implementados por essas classes. Ela promove a reutilização de código, a interoperabilidade entre classes e a flexibilidade no design de sistemas.

Exemplo:

```
// Definição da interface
public interface IVeiculo
{
    void Acelerar();
    void Parar();
}

// Implementação da interface em uma classe Carro
public class Carro : IVeiculo
{
    public void Acelerar()
    {
        Console.WriteLine("O carro está acelerando.");
    }

    public void Parar()
    {
        Console.WriteLine("O carro parou.");
    }
}
```

Pilares da POO

SENAI

Orientação a Objetos
Conceito

Orientação a Objetos
Abstração

Abstração
A POO lida com uma representação da realidade, portanto, é importante entender o que o objeto realizará dentro do sistema.

Abstração é um conceito-chave na Programação Orientada a Objetos (POO), onde se busca representar objetos do mundo real em sistemas de software.

Envolve três aspectos principais:
- a atribuição de uma identidade única ao objeto,

- a definição de suas características como propriedades;

- especificação de suas ações como métodos.

Isso simplifica a modelagem e implementação de objetos e seus comportamentos dentro do sistema.

Abstração é o processo de identificar características essenciais de um objeto e ignorar detalhes menos relevantes.
Na POO, a abstração permite modelar entidades do mundo real de forma simplificada, focando apenas nos aspectos relevantes para o problema em questão.

Exemplo
Considere o conceito de um carro. Para abstrair um carro, podemos nos concentrar em características essenciais como cor, modelo e ano, ignorando detalhes menos relevantes como o número de série do motor.

Isso nos permite criar uma representação do carro que é mais fácil de entender e manipular em nosso sistema de software.

Orientação a Objetos
Encapsulamento

Encapsulamento

Consiste em separar os aspectos externos de um objeto, os quais são acessíveis a outros objetos, dos detalhes internos de implementação do objeto, os quais permanecem escondidos dos outros objetos.

Visa ocultar os detalhes internos de uma classe e expor apenas a interface necessária para interagir com seus objetos. Isso significa que os atributos de uma classe são protegidos e acessados apenas por métodos definidos dentro da própria classe.

O uso de encapsulação evita que um programa torne-se tão interdependente que uma pequena mudança tenha grandes efeitos colaterais.

Níveis de Visibilidade

`public`: Membros públicos são acessíveis de qualquer lugar no código, dentro da classe ou do mesmo assembly (projeto) ou em assemblies externos.

`private:` Membros privados são acessíveis apenas dentro da própria classe onde são declarados. Eles permanecem ocultos para classes derivadas e assemblies externos.

`protected:` Membros protegidos são acessíveis dentro da classe e em classes derivadas, mesmo que essas classes estejam em assemblies diferentes.

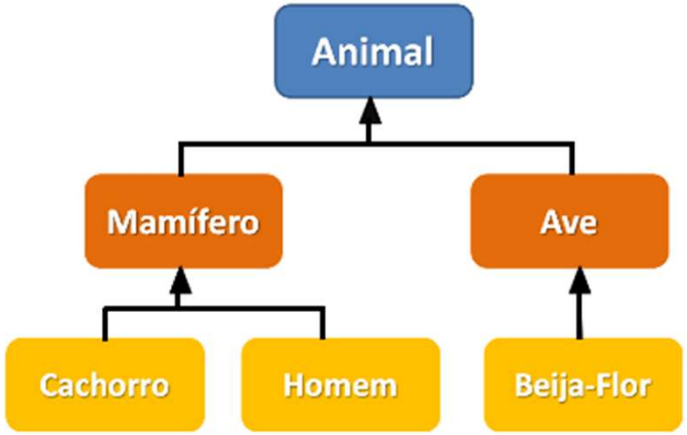
Orientação a Objetos
Herança

Herança

Permite que uma classe herde características (atributos e métodos) de outra classe.

A herança permite a reutilização de código, promovendo a extensibilidade e a modularidade do código. Ela permite que novas classes compartilhem características comuns de classes existentes, evitando a duplicação de código e facilitando a manutenção.

Por exemplo, considere uma classe "Animal" que possui métodos e atributos comuns a todos os animais, como nome e idade. Podemos criar subclasses como "Cachorro", "Gato" e "Pássaro", que herdam os atributos e métodos da classe "Animal", mas também podem ter seus próprios atributos e métodos específicos.



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.

Orientação a Objetos
Polimorfismo

Polimorfismo

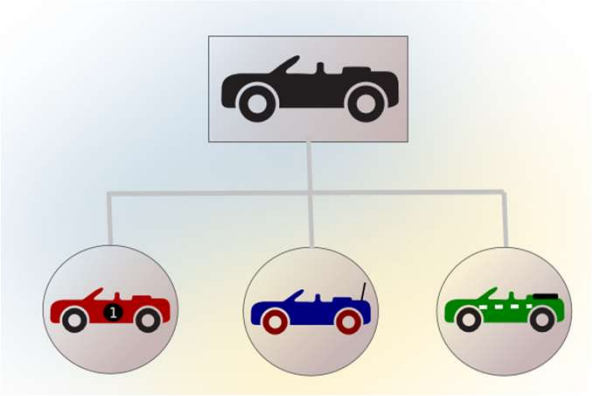
No contexto da programação orientada a objetos (POO), o polimorfismo permite que objetos de diferentes classes respondam ao mesmo método de maneiras diferentes, proporcionando flexibilidade e reutilização de código.

Isso significa que um objeto de uma classe pode se comportar como se fosse de outra classe relacionada por herança ou por implementação de uma mesma interface.

Polimorfismo Estático (Sobrecarga de Métodos): Definir múltiplos métodos com o mesmo nome, mas com diferentes parâmetros (número e/ou tipos). O compilador escolhe qual método usar com base nos argumentos fornecidos durante a chamada.

Polimorfismo Dinâmico (Sobrescrita de Métodos): Uma classe derivada pode sobrescrever um método da classe base, fornecendo uma implementação específica. Durante a execução, o método da classe do objeto em questão é chamado, mesmo que a referência seja do tipo da classe base.

O polimorfismo aumenta a flexibilidade e a reutilização do código, permitindo que o mesmo código seja aplicado a diferentes tipos de objetos, desde que esses objetos compartilhem uma relação de herança ou implementem uma mesma interface.



Orientação a Objetos
Exercícios

1. Explique a diferença entre classe e objeto na Programação Orientada a Objetos (POO). Dê exemplos para ilustrar cada conceito.

2. O que é herança e como ela é implementada? Explique o conceito de classe pai (superclasse) e classe filha (subclasse) em herança. Dê um exemplo de herança em Java com duas classes que demonstram esse conceito.

3. Explique o conceito de encapsulamento em Programação Orientada a Objetos (POO). Por que o encapsulamento é importante? Dê exemplos de como usar modificadores de acesso para implementar encapsulamento em uma classe.

4. Faça uma breve pesquisa sobre UML e explique por que ele é importante na construção de sistemas que utilizam POO. Dê um exemplo de aplicação.

5. Faça um resumo de cada um dos tópicos abordados.

Orientação a Objetos
Exercícios

Crie uma classe chamada Moto, a classe deve conter:
- 3 atributos;
- 2 métodos: 1 para mostrar os atributos e 1 para indicar que a moto está a 40km/h.

O objeto deve ser instanciado na classe que contém o Main.

Orientação a Objetos
Exercícios

Conta Bancária
Você deve criar um programa em Java que simule operações básicas de uma conta bancária. O sistema deve permitir criar uma conta, realizar depósitos, saques e exibir o saldo atual. Para isso, você precisará implementar duas classes: ContaBancaria e Main.


SENAI
DEPARTAMENTO REGIONAL
DE SÃO PAULO
www.sp.senai.br