

CODIFICAÇÃO BACK-END - SA2

SENAI

SUMÁRIO

PARA COMEÇAR	3
SITUAÇÃO-PROBLEMA	3
DESAFIO 1.....	5
SERVER-SIDE.....	6
TIPOS DE SITE.....	7
LINGUAGENS DE PROGRAMAÇÃO DEDICADAS AO SERVER-SIDE.....	12
C#	14
NESTE DESAFIO.....	21
DESAFIO 2.....	22
INTRODUÇÃO.....	23
O QUE É UM PARADIGMA?	23
PROGRAMAÇÃO PROCEDURAL.....	24
PROGRAMAÇÃO ORIENTADA A OBJETOS	24
CLASSE E OBJETO	25
criando a modelagem orientada a objetos.....	32
CONCEITOS DE OO	34
TRATAMENTO DE ERROS	42
DEPURAÇÃO.....	43
NESTE DESAFIO.....	43
DESAFIO 3.....	44
DOCUMENTAÇÃO.....	44
README.MD	48
GITHUB	54
NESTE DESAFIO.....	55
PARA CONCLUIR.....	55
REFERÊNCIAS	57
CRÉDITOS.....	58

PARA COMEÇAR

Este material aborda conceitos relacionados à codificação de sistemas web server-side (Back-End), utilizando a programação orientada a objetos para atender às regras de negócio estabelecidas.

Esperamos que você desenvolva, no decorrer de seus estudos, as seguintes capacidades:

- Reconhecer as linguagens de programação (C#) dedicadas ao server-side;
- Reconhecer os diferentes tipos e formatos de dados e de arquivo;
- Aplicar técnicas de conversão e de manipulação de dados e de arquivos;
- Aplicar técnicas e métodos de desenvolvimento, conforme a linguagem de programação empregada.

Para desenvolver tais capacidades, você deverá estudar os seguintes temas:

- Linguagem de programação
- Manipulação e conversão de arquivos
- Framework
- Modelagem
- IDE
- Documentação

O estudo desses temas será necessário para que você resolva a situação-problema a seguir. Então, avance para conhecê-la.

SITUAÇÃO-PROBLEMA

A Troppo é uma empresa de desenvolvimento de software que utiliza os melhores recursos da área em suas aplicações. Robustez, performance e escalabilidade são alguns dos termos valorizados e utilizados para desenvolver os seus projetos.

A ClientLab é uma empresa focada na gestão de clientes e, atualmente, mantém a sua base de registro em papel e não possui nenhuma automatização ou tecnologia que facilite o processo de busca de clientes e/ou registro de suas atividades. Tendo em vista o cenário atual, cada vez mais tecnológico, a ClientLab contratou a Troppo para realizar o desenvolvimento de seu novo sistema que realizará essas atividades.

Assim, o contrato estabelece que a Troppo realizará a programação de um sistema de cadastro de clientes customizado, atendendo às seguintes características:

- o sistema de clientes deverá armazenar os cadastros das pessoas físicas e jurídicas;
- o cadastro das pessoas físicas é feito com os seguintes dados: nome, CPF e data de nascimento;
- o cadastro das pessoas jurídicas é feito com os seguintes dados: nome e CNPJ;
- ambos devem possuir a opção para pagar impostos;
- o sistema deve armazenar os registros em arquivos.

Você deverá se atentar às regras estabelecidas, tratando os valores não permitidos inseridos pelos usuários. Para isso, você deve resolver três desafios.

Clique nos botões a seguir para conhecê-los:

Desafio 1

Aplicar linguagem de programação para atender às regras de negócio estabelecidas.

Desafio 2

Aplicar programação orientada a objetos na construção do sistema.

Desafio 3

Utilizar linguagem de marcação para a apresentação dos recursos disponibilizados na situação-problema.

DESAFIO 1

Nesta etapa, você deverá resolver o desafio 1:

- Aplicar linguagem de programação para atender às regras de negócio estabelecidas.

Para isso, você estudará os seguintes conteúdos:

- Linguagem de programação
 - Variáveis e constantes
 - Operadores
 - Laços de repetição condicionais
 - Classes
 - Função
 - Bibliotecas
- Manipulação de arquivos
 - Conversão de arquivos



SERVER-SIDE



Fonte: Unsplash

O termo **server-side** é relacionado às ações que acontecem do lado do servidor. No lado do usuário, ou cliente-side, as linguagens são processadas pelo navegador.

Servidor é um computador que fornece recursos a clientes e é utilizado em tudo que envolva internet ou armazenamento de dados. Um servidor local, por exemplo, é usado para o compartilhamento de arquivo, ao passo que um servidor web armazena os documentos de sites, como HTML, CSS e JavaScript, processando requisições e devolvendo informações ao cliente.

A programação Back-End permite enviar informações personalizadas, fornecendo uma melhor experiência ao usuário. Isso ocorre porque os dados são armazenados em um banco de dados e a programação *server-side* dinamiza os dados retornados. Por exemplo, você pode realizar operações permitidas para o usuário e publicar posts e tweets ao acessar sites como Facebook ou Twitter.

Além da personalização de conteúdo, a programação *server-side* pode, também, fornecer restrição de acessos, notificações (SMS, por exemplo), entre outras funções. Antes de abordar as aplicações, vamos conhecer alguns processos e conceitos importantes.

HTTP E URL

O HTTP, ou protocolo de transferência de hipertexto, permite a troca de dados na web (cliente-servidor), como vídeos, imagens e documentos HTML.

A solicitação HTTP feita contém a URL do destinatário, assim, é definida a ação necessária e as informações adicionais que podem ser incluídas na solicitação. Os métodos são importantes justamente em razão de possuírem a identificação da ação necessária. Por exemplo, podemos obter um recurso (para visualizar a imagem de perfil de um colega no Instagram), adicionar uma nova imagem (postar), atualizar ou deletar uma imagem já cadastrada em nosso perfil.



Fonte: Pixabay

Após a solicitação ser feita, o navegador aguarda por uma resposta (response) do servidor, que identifica por meio do status (um código), existente na mensagem, se a solicitação foi ou não bem-sucedida. Essa informação é importante para identificar se a requisição HTTP foi concluída corretamente.



As respostas são categorizadas em 5 grupos:

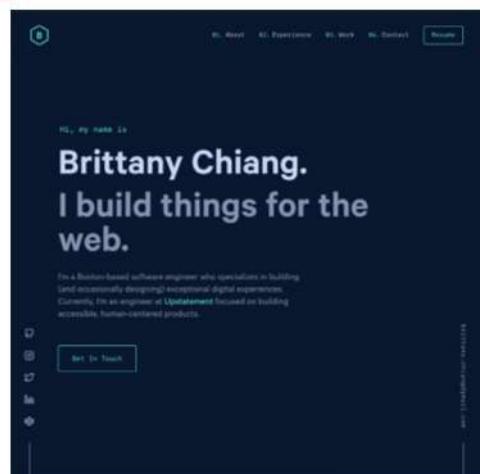
- Respostas de informação (100-199);
- Respostas de sucesso (200-299);
- Redirecionamentos (300-399);
- Erros do cliente (400-499);
- Erros do servidor (500-599).

TIPOS DE SITE

Dependendo da codificação Back-End, os sites podem ser estáticos ou dinâmicos, alterando a comunicação entre servidor e cliente.

SITE ESTÁTICO

Um site estático retorna sempre o mesmo conteúdo. Além disso, não possui ferramentas de gerenciamento incorporadas a ele. São modelos

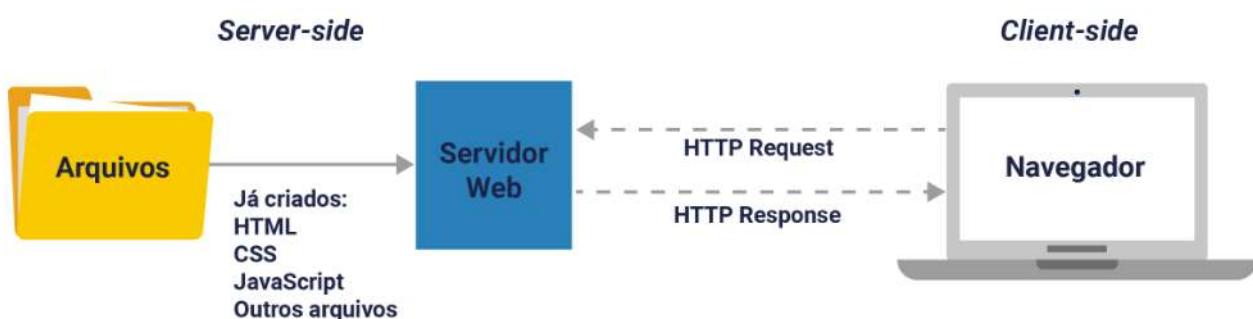


Fonte: <https://brittanychiang.com/>

utilizados para consultar informações sobre as empresas, como serviços e produtos, e não há a necessidade de atualizar o conteúdo da página com tanta frequência. Veja um exemplo ao lado.

Caso deva ser feita uma alteração na página, uma alteração no código-fonte (usualmente HTML, CSS e JavaScript) precisará ser realizada.

Sem reformulação no código-fonte, não ocorrerão mudanças. Quando o usuário solicita uma página, o navegador web envia a solicitação para o servidor, que recupera os documentos e os retorna, com o status, ao navegador para serem renderizados, conforme mostra o esquema abaixo.



Fonte: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Introduction

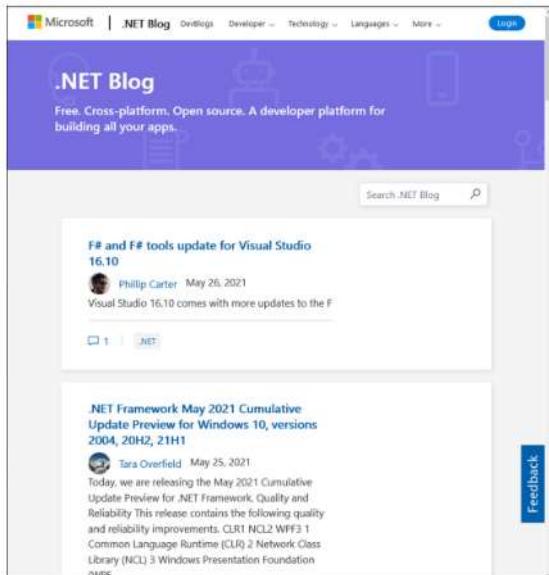
SITE DINÂMICO

Em um site dinâmico, todo ou parte do conteúdo é gerado dinamicamente. A inclusão, a alteração ou a exclusão de uma informação podem ser feitas por meio de um sistema de gerenciamento de conteúdo.

Portais de notícias, redes sociais, bancos de imagens e sites com muito conteúdo, em geral, costumam ser do tipo dinâmico. Veja o exemplo da imagem .NET Blog.

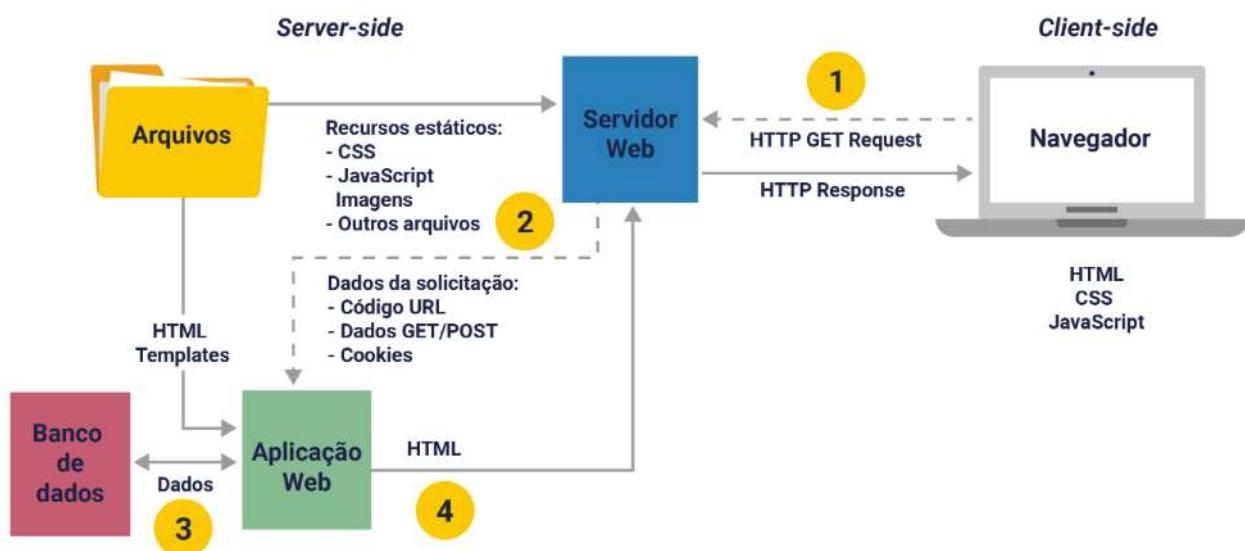
Com a criação de conteúdos dinâmicos, muitas aplicações web surgiram para atender aos diversos tipos de mercado e de contextos que a web exige.

Para construir a dinâmica e as regras de negócios que os sistemas exigem, construímos aplicações Back-End, que podem ler informações dos bancos de dados e organizá-las dinamicamente. É muito provável que você entre em contato com pelo menos uma dezena delas por dia entre aplicações bancárias, blogs e redes sociais.



Fonte: <https://devblogs.microsoft.com/dotnet/>

Ao solicitar uma informação ao servidor, um site dinâmico identifica que uma parte do conteúdo será gerado dinamicamente.



Adaptado de: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Introduction

1. Usuário --> Cliente

O usuário entra com uma URL em seu navegador, por exemplo: <http://www.instagram.com>.

2. Cliente --> Servidor

O cliente, no caso o navegador do usuário, realiza a requisição (request) da página

ao servidor do site do Instagram.

3. Servidor --> Cliente

O servidor recebe a requisição, obtém a página e a envia ao cliente (*response*).

4. Cliente --> Usuário

O cliente renderiza os recursos recebidos pelo servidor e os exibe ao usuário.

FRAMEWORK

Para otimizar o desenvolvimento tanto de sites dinâmicos quanto estáticos, usamos framework.

Framework é um conjunto de *bibliotecas*¹ utilizado para desenvolver aplicações com uma estrutura pré-definida, podendo simplificar o processo de desenvolvimento e de ganho de performance.



Neste conteúdo, utilizaremos a linguagem de programação C# (pronuncia-se "see sharp") e o correspondente framework .NET (pronuncia-se "dotnet").

O C# foi lançado em 2002, com o Visual Studio .NET e buscava ser uma linguagem simples, moderna, de uso geral e orientada a objetos.

O .NET é uma plataforma de desenvolvimento de software cujo tempo de vida e maturidade de desenvolvimento a tornam uma das plataformas mais utilizadas por empresas. Além de compatível com o C#, o .NET oferece suporte a outras linguagens, como o F# e o Visual Basic.

O .NET fornece a base para a criação de vários tipos de aplicativos, principalmente, **aplicativos desktop, aplicativos móveis, aplicativos web** ou jogos, e tem suporte para os sistemas operacionais Windows, macOS e Linux.

¹ Conjunto de subprogramas independentes que permitem o compartilhamento, o uso modular e a personalização de códigos.



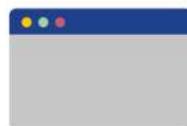
Aplicativos desktop

Os aplicativos desktop caracterizam-se como programas executados pelo usuário em seu sistema operacional sem fazer uso de um navegador web.



Aplicativos móveis

Os aplicativos móveis são caracterizados por serem instalados e utilizados em aparelhos celulares que funcionam com os sistemas operacionais Android ou iOS, estando disponíveis nas lojas Google Play e Apple Store, respectivamente.



Aplicativos web

Os aplicativos web são armazenados em um servidor e podem responder tanto a conteúdos estáticos, que possuem sempre o mesmo retorno, quanto a conteúdos dinâmicos, por meio de um conjunto de informações recuperadas de uma consulta ao banco de dados.

O .NET é distribuído em dois pacotes. Clique nas abas para conhecê-los.

SDK

O Software Development Kit (SDK) possui o que é necessário para criarmos uma aplicação em ambiente de desenvolvimento, a partir do .NET, viabilizando, assim, o processo de compilação e de execução da criação de nossos programas. O SDK possui também o Runtime.

Runtime

Caso você tenha a aplicação pronta para ser executada, é possível utilizar somente o Runtime para fazê-la funcionar.

Saiba mais

Para saber mais sobre o .NET, acesse o link: <https://docs.microsoft.com/pt-br/dotnet/core/introduction>



LINGUAGENS DE PROGRAMAÇÃO DEDICADAS AO SERVER-SIDE

Há diversas maneiras de classificar as linguagens de programação. Podemos, por exemplo, classificá-las em: compilada ou interpretada; de alto nível ou de baixo nível etc. Agora, estudaremos a classificação feita por tipo de paradigmas.

Em programação, um paradigma é uma abordagem utilizada para resolver problemas a partir de técnicas e de ferramentas disponíveis e que podem ser aplicadas a diferentes linguagens. Paradigmas são, portanto, abordagens diferentes para a criação de nossas aplicações.



Fonte: Pexels

Uma linguagem de programação pode aplicar paradigmas diferentes. O importante é compreender qual pode ser o paradigma mais adequado à construção de seu projeto.

Veja a definição resumida de alguns paradigmas de programação:

Linguagem estruturada

A programação estruturada é uma técnica que pode ser utilizada em diferentes linguagens, permitindo a construção de programas em um número restrito de estruturas. As três estruturas são: de controle de repetição, de condição e de sequência.

Orientada a objetos

O paradigma de programação orientado a objetos (POO ou, em inglês, OOP) surgiu como uma alternativa para aproximar a representação dos itens do mundo real aos do mundo da programação; baseia-se em classes e objetos.

Programação funcional

A programação funcional é o processo de construção de sistemas baseado em funções puras, ou seja, evitando-se o compartilhamento de estado. A partir de um dado de entrada, constrói-se uma saída esperada.

Programação imperativa

A programação imperativa é baseada em instruções e comandos, ou seja, são estabelecidas ordens sob as quais o sistema deve operar (indicando-se como e o que o sistema deve realizar).

Compare as características de algumas linguagens de programação.

Nome da linguagem	Paradigma	Paradigma de programação	Implementação	Aplicações
C#	Multi	Orientação a objetos; funcional	Compilada	Aplicações web, mobile e desktop
Java	Multi	Orientação a objetos; imperativa	Compilada	Muito utilizada em aplicações mobile (Android)
Elixir	Multi	Orientada a processos; funcional; concorrente; distribuída	Compilada	Utilizada em sistemas distribuídos e em APIs web
Clojure	Único	Programação funcional	Compilada	Big data e dados

Saiba mais

Para saber mais sobre paradigmas de programação, acesse o link:
<https://docs.microsoft.com/pt-br/dotnet/standard/linq/functional-vs-imperative-programming>



C#



C# é uma linguagem de programação suportada pelo .NET; é a principal linguagem da Microsoft e possui as seguintes características:

- linguagem de alto *nível*²;
- *orientada a objetos*³;
- de tipo seguro (fortemente *tipada*⁴);
- *case-sensitive*⁵;
- *compilada*⁶.

Por ser uma linguagem compilada, o C# fornece a visualização dos erros em tempo de desenvolvimento e, em caso de falha, o código não é executado.

O C# possui maturidade no mercado (a linguagem passou por testes, validações e atualizações ao longo do tempo), sendo uma das linguagens mais utilizadas por empresas no mundo inteiro. A orientação a objetos permite a reutilização em outros cenários e contextos (inclusive, o C# te orienta a seguir essa linha). Todos os conhecimentos e conceitos adquiridos neste desafio podem ser reaplicados em outras linguagens, como o Java.

Saiba mais



O processo de traduzir uma linguagem de alto nível, como o C#, para uma linguagem de máquina é denominado compilação. Para saber mais a respeito desse processo, acesse: <https://docs.microsoft.com/pt-br/dotnet/standard/managed-code>.

2 A grosso modo, aproxima-se mais da linguagem humana do que a linguagem de baixo nível. A linguagem de baixo nível é a linguagem das máquinas e é composta apenas de números binários.

3 Método da programação utilizado para estruturar o código em pedaços simples e que viabiliza a reutilização de um código.

4 Linguagem cujas variáveis têm um tipo específico, geralmente, definido na declaração.

5 Que faz a diferenciação entre maiúsculas e minúsculas.

6 Linguagem cujo código-fonte pode ser traduzido pelo sistema operacional ou processado para a linguagem de baixo nível, por meio de um programa de computador denominado compilador.

AMBIENTE DE DESENVOLVIMENTO

Para poder codificar com o .NET e a C# é necessário preparar o ambiente de desenvolvimento.

O ambiente de desenvolvimento é composto de ferramentas e utilitários necessários para o desenvolvimento de novos programas. Você deverá instalar o SDK do .NET, na versão 5, para ter acesso às ferramentas de criação e de execução do projeto no .NET.

PDF

Acesse seu material complementar e veja o tutorial de instalação do SDK.

Arquivo: 01_instalando_SDK_DOTNET.pdf

Interface de linha de comando

A partir da instalação do SDK, você terá acesso a uma interface que adiciona comandos ao nosso terminal, o que permite executar operações. Essa interface é chamada **dotnet cli** (Command Line Interface) e permite:

- criar projetos de diferentes tipos (como console, web e desktop);
- verificar a versão instalada (como fizemos no passo anterior);
- utilizar a pasta de referência dotnet.

FERRAMENTAS

Há duas opções de ferramentas recomendadas para a utilização em seus projetos: VSCode e Microsoft Visual Studio, mas existem outras (como o ReSharper, da IntelliJ).

O **Visual Studio Code (VSCode)** é um editor de código leve que tem ganhado popularidade por permitir a criação de diversas áreas, como web, desktop e mobile. Para adicionar o suporte à linguagem C#, você deverá adicionar a extensão no VSCode.

O *ambiente de desenvolvimento integrado*⁷ da Microsoft é chamado **Microsoft Visual Studio** ou, simplesmente, Visual Studio. Trata-se de uma IDE gratuita.

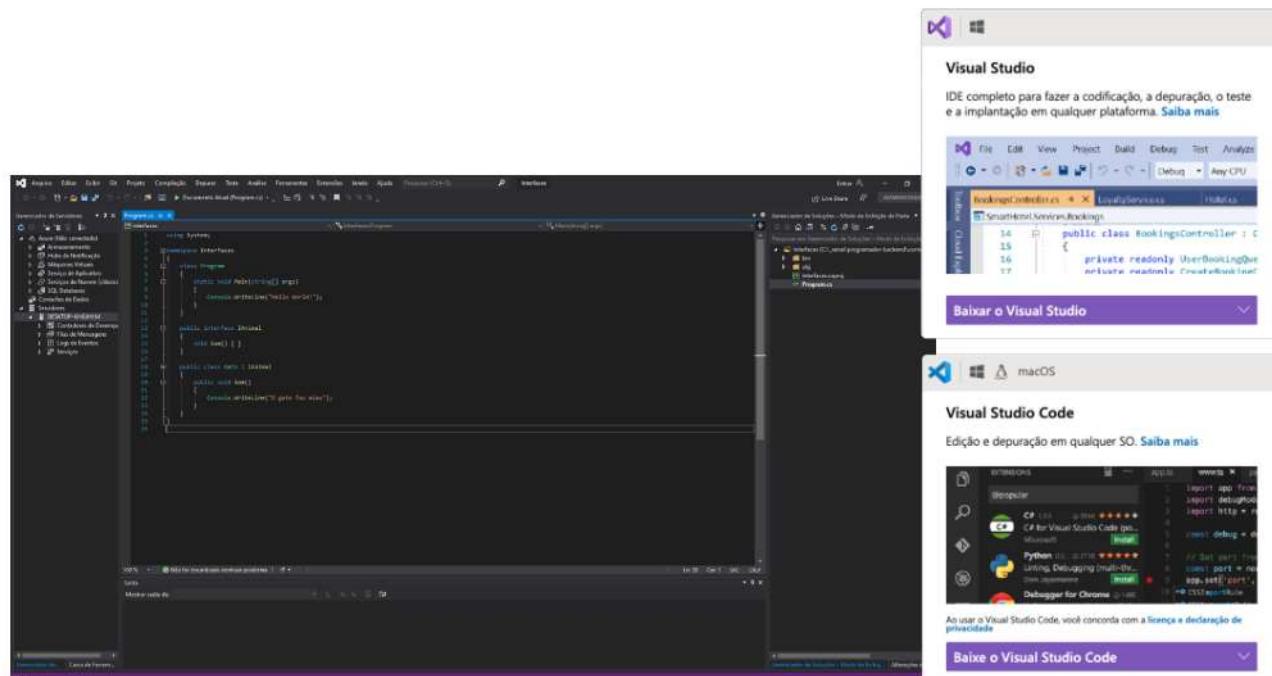
PDF

Para acessar o tutorial de instalação do VSCode, verifique seu material complementar
Arquivo: 02_instalando_VSCode.pdf

Para o tutorial da IDE do Microsoft Visual Studio, verifique seu material complementar
Arquivo: 03_instalando_Visual_Studio.pdf

Você pode acompanhar o roteiro de instalação ou pode encontrar mais informações e detalhes acessando o link: <https://visualstudio.microsoft.com/pt-br/vs/community/>

Essa imagem é de um projeto em desenvolvimento aberto na interface da IDE Visual Studio.



ESTRUTURA DO PROJETO

Com a interface de linha de comando instalada, você poderá ter acesso a comandos que permitem criar projetos, testá-los e executá-los.

⁷ Ambiente de Desenvolvimento Integrado ou IDE (Integrated Development Environment) é um programa que reúne ferramentas de desenvolvimento com utilitários integrados, que conta, entre outras coisas, com destaque de sintaxe e recursos visuais de identificação.

O comando **dotnet new** cria um projeto baseado em um modelo de sua escolha: aplicativo de console, aplicativo web asp.net core, projetos de testes e muitos outros.



PDF

Acompanhe o passo a passo para criar um projeto com .NET
Acesse o seu material complementar.

Arquivo: 04_criando_projeto_DOTNET.pdf

Para acessar o tutorial de como abrir o projeto no VSCode, verifique seu material complementar

Arquivo: 05_abrindo_projeto_VSCode.pdf

Com a compreensão sobre a criação do projeto e de sua estrutura-base, você estudará conceitos básicos da sintaxe do C#, como tipos de dados e variáveis, e aprenderá a alterar o fluxo de execução de uma aplicação em seu código.

LÓGICA DE PROGRAMAÇÃO

Antes de estudarmos a sintaxe básica do C#, retomaremos alguns conceitos básicos de lógica de programação.

PDF

Para retomar os conceitos, acesse o material complementar da lista a seguir:

- Variáveis, constantes, dados e palavras-chave. **Arquivo:** 06_variaveis_constants.pdf
- Comentários. **Arquivo:** 07_comentario.pdf
- Operadores. **Arquivo:** 08_operadores.pdf
- Estruturas de controle. **Arquivo:** 09_estruturas_controle.pdf
- Estruturas de repetição. **Arquivo:** 10_estruturas_repeticao.pdf
- Arrays. **Arquivo:** 11_arrays.pdf
- Funções e bibliotecas. **Arquivo:** 12_funcoes_biblioteca.pdf
- Exemplo com código comentado. **Arquivo:** 13_exemplo_Logica_Programacao.pdf

TESTANDO A BIBLIOTECA

Durante o desenvolvimento do projeto, são criados diversos arquivos com finalidades específicas, como guardar dados, converter informações de uma linguagem para outra, fazer a conexão entre módulos do framework, entre outros.



Para cada tipo de arquivo, é necessário utilizar programas específicos para ler certos formatos de arquivos.

Os formatos indicam como eles são armazenados, para serem consultados e visualizados posteriormente. Por exemplo: os arquivos de imagem existentes em seu computador possuem uma variação de .png, .jpg ou .jpeg, podendo ser abertos com o software leitor de fotos do sistema operacional Windows, para serem visualizados, ou com o navegador instalado (Firefox, Chrome ou Edge).

Os arquivos de texto, por sua vez, podem conter a extensão .txt (possuem uma limitação de edição e de formatação), ou .doc e .docx, próprios da Microsoft, sendo abertos, visualizados e editados com o programa Word.

Além dos softwares específicos de edição de texto, podemos usar bibliotecas para manipular um arquivo .txt. As operações de leitura e de escrita nos arquivos são comuns para armazenar dados e realizar a leitura de informações de um arquivo existente em disco e, por isso, essas operações estão presentes nas bibliotecas.

Os módulos do .NET agilizam nosso desenvolvimento, pois permitem o acesso a recursos do sistema, como leitura e gravação em arquivos com o formato .txt.

Agora vamos criar uma pasta chamada Arquivos para testar a manipulação de um arquivo de texto. Em seguida, vamos criar o arquivo.txt e realizar as operações de leitura e escrita nele, usando uma biblioteca do .NET.

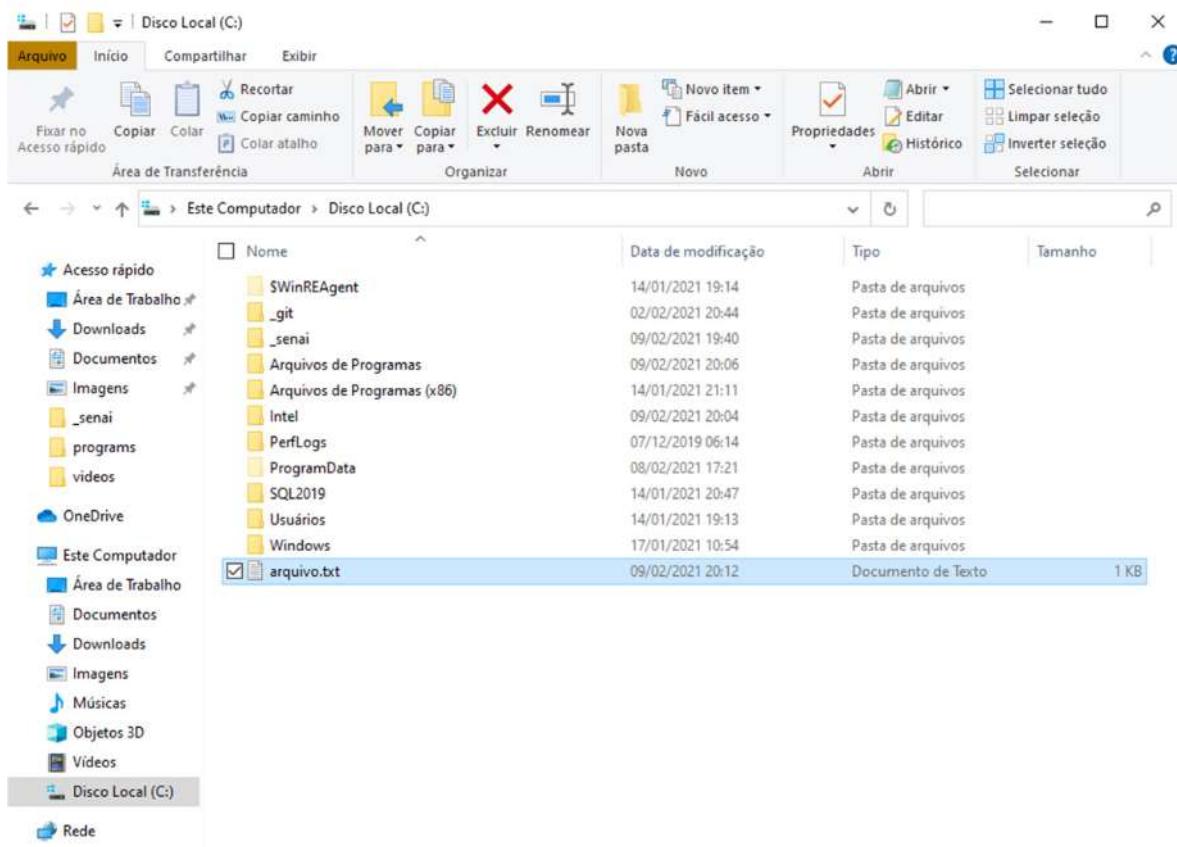
PDF

Acesse seu material complementar e acompanhe o passo a passo para criar o projeto Arquivos.
Arquivo: 14_criando_projetos_Arquivos.pdf

Manipulação de arquivos

Agora vamos ver como manipular um arquivo de texto usando uma biblioteca do .NET.

Em seu computador, crie um arquivo chamado arquivo.txt, no qual realizaremos algumas operações. No exemplo, o arquivo.txt foi criado no disco local (geralmente identificado como C:\), como mostra a imagem a seguir.



Acompanhe no código comentado a seguir a biblioteca e os métodos para manipular o arquivo.txt.

O **using** contém as importações de bibliotecas. O **namespace** define uma separação lógica de pastas.

O **namespace System.IO** contém classes que permitem realizar a leitura e a gravação em arquivos, bem como fornecem acesso ao diretório de arquivos. Inclua **System.IO** como **using**, que contém os tipos mencionados, como mostra a imagem a seguir.

Na linha 10, é definido o local de seu computador em que o seu arquivo será salvo (local de referência).

Com o **StreamWriter(caminhoDoArquivo)**, um novo **StreamWriter** é inicializado com o nome do arquivo.

O **stream.WriteLine("")** permite gravar uma cadeia de caracteres.

O **stream.Close()** fecha o fluxo atual do **StreamWriter**.

Ao utilizar o **using**, a execução será finalizada ao final do bloco dessa instrução, não sendo necessário fechar e finalizar a utilização desse recurso.

O **StreamReader.ReadLine()** permite realizar a leitura das informações desse arquivo .txt.

The screenshot shows a Visual Studio code editor window with the following C# code:

```
1  using System;
2  // namespace System.IO - contém tipos que permitem a leitura e escrita em arquivos
3  using System.IO;
4
5  namespace Arquivos
6  {
7      class Program
8      {
9          static void Main(string[] args)
10         {
11             // caminho do arquivo
12             string caminhoDoArquivo = "C:\\arquivo.txt";
13             // utilizado para escrever data em um arquivo
14             var stream = new StreamWriter(caminhoDoArquivo);
15             // escreve uma linha no arquivo
16             stream.WriteLine("Texto salvo com sucesso.");
17             // Fecha o objeto atual e o fluxo adjacente
18             stream.Close();
19             // o using finaliza a instrução no final do bloco de execução
20             // streamreader para ler um arquivo
21             using (var reader = new StreamReader(caminhoDoArquivo))
22             {
23                 // lê a linha e retorna a string
24                 Console.WriteLine(reader.ReadLine());
25             }
26         }
27     }
28 }
29
```

NESTE DESAFIO...

CODIFICAÇÃO BACK-END | DESAFIO 01



Você estudou que servidor web armazena os documentos de sites, como HTML, CSS e JavaScript, processando requisições e devolvendo informações ao cliente, no lado do usuário. E o que permite a comunicação entre cliente-servidor é protocolo HTTP.

Também conheceu a diferença entre site estáticos e dinâmico e as regras de negócios que os sistemas exigem, que são construídas por meio das aplicações de Back-End e realizadas por meio de linguagens de programação, como a linguagem orientada a objetos C#.

Além disso, você estudou que o .NET é framework compatível com o C# e disponibiliza uma série de recursos que podem ser utilizados para realizar a gravação e a leitura de arquivos, além de disponibilizar outros recursos que facilitam o processo de desenvolvimento.

NO PRÓXIMO DESAFIO...

Você estudará como aplicar programação orientada a objetos para o desenvolvimento do sistema.

DESAFIO 2

No desafio 1, você estudou a aplicação da linguagem de programação C#, a plataforma .NET e algumas funcionalidades, como separar o código em funções menores para o reaproveitamento e a leitura/escrita em arquivos.

Neste desafio, você deverá:

- Aplicar programação orientada a objetos para o desenvolvimento do sistema.

Para isso, você estudará os seguintes conteúdos:

Frameworks.

- Modelagem.
- Padrões de desenvolvimento de interface.
- Classes de elementos gráficos.
- Tipos de aplicação.
- Propriedades dos objetos.
- Depuração.
- Configurações.



INTRODUÇÃO

As linguagens de programação adotam diferentes abordagens para resolver um problema. Uma mesma linguagem pode possuir diferentes formas, apresentando certo estilo geralmente chamado de "paradigma de programação". No mercado de trabalho, criamos soluções por meio de linha de código.

O C# fornece suporte para a linguagem de programação orientada a objetos e apresenta as seguintes vantagens:



Importante!

Para este desafio, você deverá seguir as seguintes orientações:

- Abra o terminal de comando em sua pasta de preferência.
- Crie um projeto de console com o comando dotnet new console -o OOP para realizar as atividades práticas deste desafio.
- Abra o VSCode.
- Na sequência, abra a pasta OOP no programa.



O QUE É UM PARADIGMA?

Em programação, um paradigma é uma abordagem para resolver problemas por meio de técnicas e ferramentas disponíveis e aplicáveis a diferentes linguagens. São, portanto, abordagens diferentes para a criação de aplicações.

Os principais paradigmas de programação basicamente pertencem a dois grupos, **imperativos** ou **declarativos**. Dentro dos paradigmas de programação do tipo imperativo temos:

- Programação procedural;
- Programação orientada ao objetos.

Siga em frente e conheça cada um deles.



PROGRAMAÇÃO PROCEDURAL

A programação procedural foi uma das abordagens mais utilizadas durante muito tempo. Ela é construída por um conjunto de procedimentos (funções) programados e utilizados em várias partes do código. Essas funções, descritas no primeiro desafio, são criadas e agrupadas em uma unidade para realizar uma instrução ou para estruturar um bloco de funções, que são criadas em vários blocos da aplicação para, assim, solucionar um problema maior.

Essas funções não contêm estado; uma função pode chamar outra e é difícil identificar a dependência entre elas. Portanto, é importante identificar se uma função existente é capaz de solucionar um problema que você precisa resolver.

A seguir, conheça um exemplo de função:

1. Funcao a () {}
2. Funcao b () {
3. a();
4. }

PROGRAMAÇÃO ORIENTADA A OBJETOS

O paradigma de programação orientada a objetos (POO ou OOP, em inglês) surgiu como uma alternativa para aproximar a representação dos itens do mundo real para o mundo da programação. Esse paradigma se baseia em classes e *objetos*⁸ e é

⁸ Trata-se da representação de algo que existe no mundo real e que possui significado para o software, como um produto, uma pessoa ou um aluno.

amplamente utilizado para estruturar projetos em pedaços reutilizáveis de código.

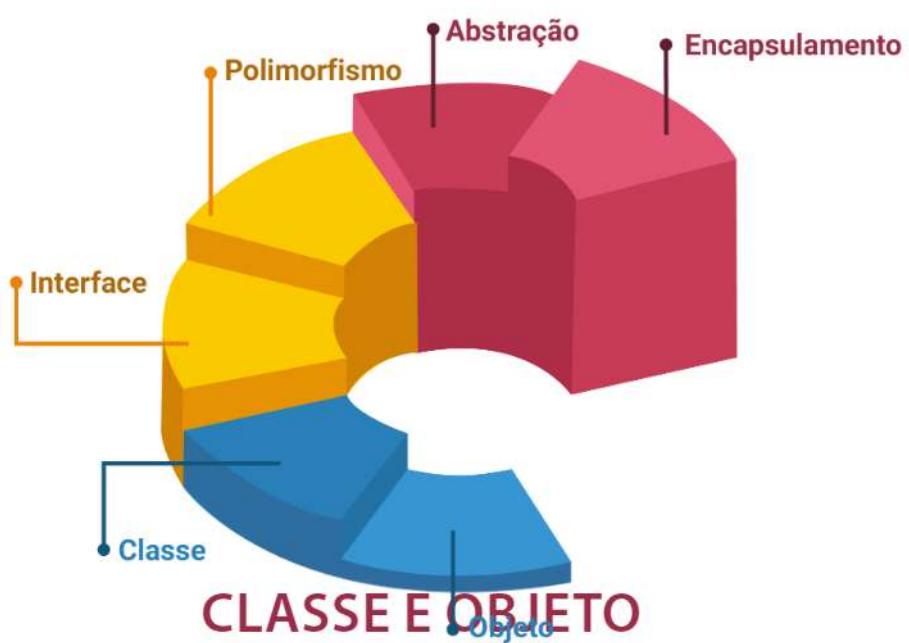
Dica!

AchaveparasermaisprodutivoemPOOétnarcadaobjetoresponsável pela realização de um conjunto de tarefas relacionadas. Se um objeto depender de uma tarefa que não seja de sua responsabilidade, ele precisará ter acesso a um outro objeto cujas responsabilidades incluem essa tarefa. Assim, os objetos trocam mensagens entre si para solucionar um problema complexo.



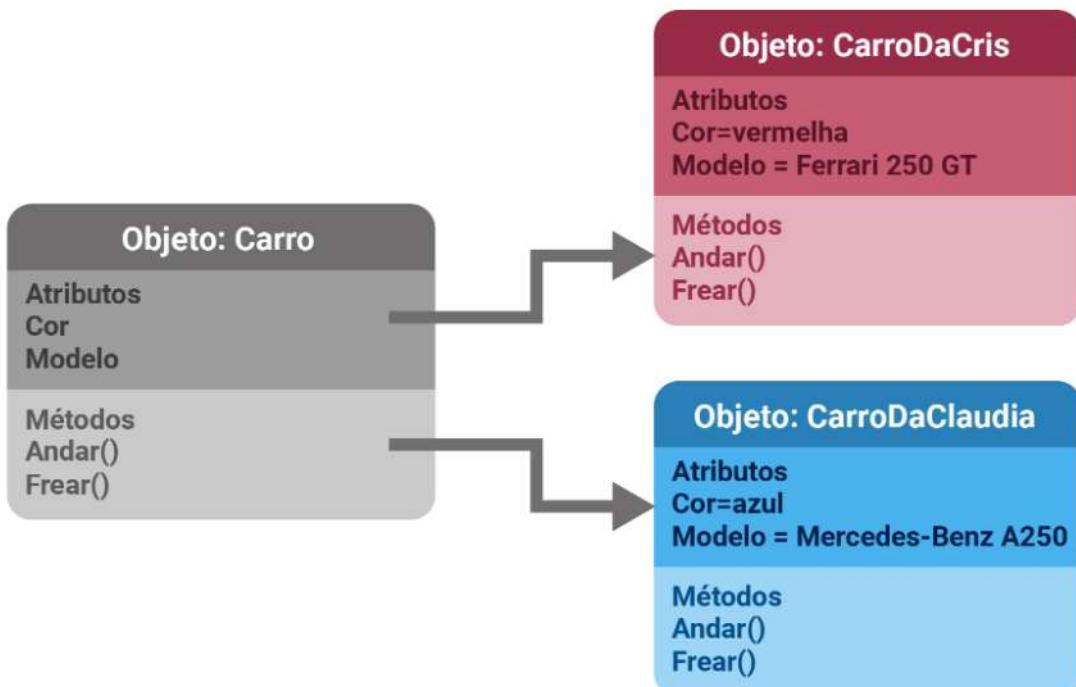
Na orientação a objetos, cada objeto deve ter sua própria responsabilidade. Se na programação procedural tivermos funções espalhadas como `calculaDescontoEmFolha()` para um determinado cargo, na programação orientada a objetos teríamos um funcionário que agrupa essa funcionalidade dentro do seu contexto.

OOP tem blocos de construção principais que são, objetos, classe, interface, polimorfismo, abstração e encapsulamento, conforme mostra o infográfico a seguir:



Geralmente, a classe é descrita como o modelo ou a forma que define o objeto. Para construir os objetos, devemos estabelecer quais serão suas características (atributos) e seus *métodos*⁹. Inúmeros objetos podem ser construídos a partir das classes.

A seguir, conheça a representação da classe **Carro** e de dois objetos criados a partir do modelo definido (entre propriedades e métodos).



A maneira correta de declarar uma classe em C# é feita utilizando-se a palavra reservada `class` seguida do nome da classe. Analise os exemplos a seguir.

Código representando a classe Carro:

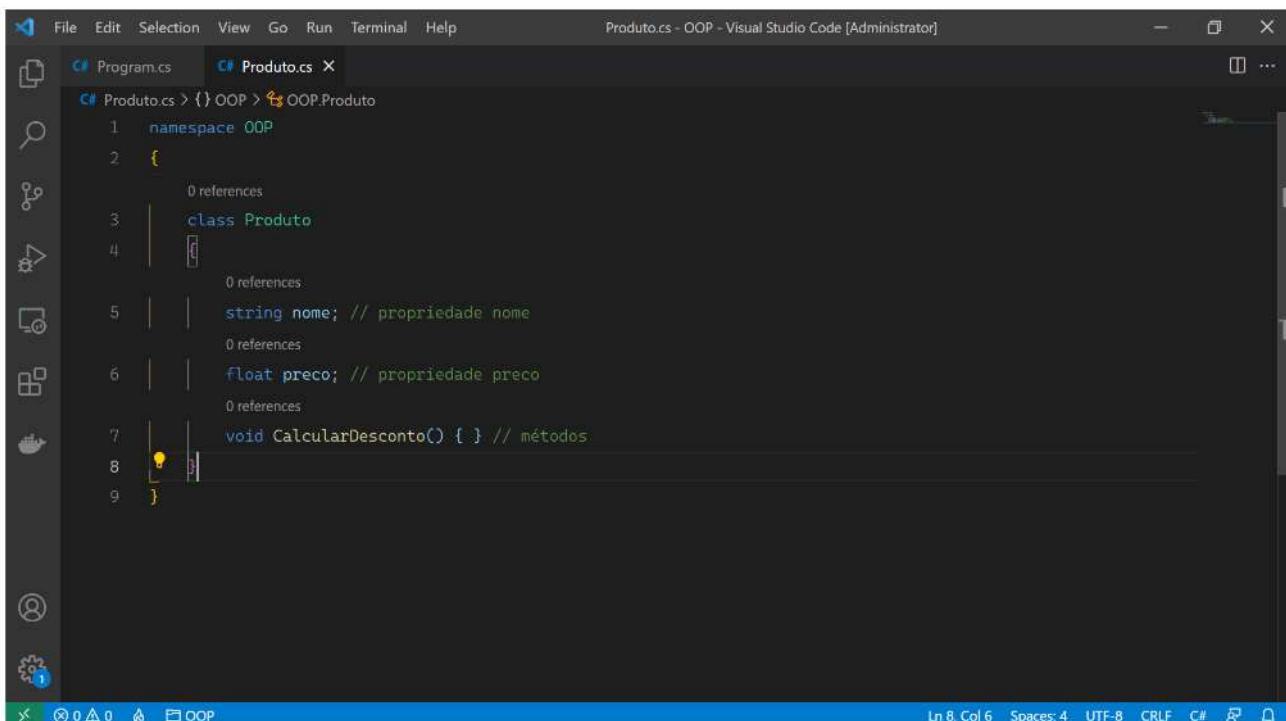
```
1. class Carro
2. {
3.     string cor; //propriedade cor
4.     string modelo; //propriedade modelo
5.     void Andar() { } //métodos
6.     void Frear() { } //métodos
7. }
```

⁹ Trata-se de uma função que faz parte de uma classe e que representa um comportamento.

Código representando a classe Produto:

```
1. class Produto
2. {
3.     string nome; //propriedade nome
4.     float preco; //propriedade preço
5.     void CalcularDesconto() { } //métodos
6. }
```

O código da classe **Produto**, por convenção, é criado dentro de um arquivo de mesmo nome (Produto.cs).



```
File Edit Selection View Go Run Terminal Help
Produo.cs - OOP - Visual Studio Code [Administrator]
Program.cs  Produto.cs
C# Produto.cs > {} OOP > OOP.Produto
1 namespace OOP
2 {
3     class Produto
4     {
5         string nome; // propriedade nome
6         float preco; // propriedade preço
7         void CalcularDesconto() { } // métodos
8     }
9 }
```

Importante!

O arquivo de extensão .cs é um arquivo de código escrito em C#.



Dentro da classe **Produto** são armazenadas informações que a descrevem e comportamentos que a compõem. Um princípio da **POO** que facilita o processo de desenvolvimento é que **uma classe pode ser formada a partir de outras**.

Recomendação

Uma regra simples de ser utilizada na identificação e análise de um problema é a seguinte: procurar por substantivos (nomes) para as classes e, para os **métodos**, procurar por verbos (ações).

A título de exemplo, podemos pensar em um sistema de compras online, que utiliza alguns dos substantivos a seguir:

- item;
- produto;
- endereço;
- pagamento.

Esses nomes podem te ajudar na criação de classes, ao passo que os verbos (como: aplicar desconto ou adicionar em estoque) poderão ser métodos da classe, uma vez que são ações realizáveis pelos objetos. Perceba, por fim, que eles representam itens do mundo real.

A **classe Produto** é a classe modelo. Para criarmos um objeto (ou, como chamamos, uma instância da classe), utilizamos o operador *new*.

```

File Edit Selection View Go Run Terminal Help
Program.cs - OOP - Visual Studio Code [Administrator]
Explorer (Ctrl+Shift+E) ...
File Edit Selection View Go Run Terminal Help
Program.cs > {} OOP > OOP.Program > Main(string[] args)
1 using System;
2
3 namespace OOP
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Produto p = new Produto();
10        }
11    }
12 }

Ln 9, Col 11  Spaces: 4  UTF-8 with BOM  CRLF  ⌂  ⌂  ⌂

Produto.cs > {} OOP
namespace OOP
{
    class Produto
    {
        public string nome; // propriedade nome
        public float preco; // propriedade preco
        public void CalcularDesconto() { } // métodos
    }
}
Ln 9, Col 11  Spaces: 4  UTF-8 with BOM  CRLF  ⌂  ⌂  ⌂

```

Listas

No seu dia a dia, ao navegar em um site, visualizar seu blog favorito ou acessar as suas redes sociais, você provavelmente terá acesso a uma lista de itens apresentadas na tela.

A seguir, temos como exemplo o site da Editora SENAI, no qual você poderá conferir uma lista de obras que a editora possui em seu catálogo.

The screenshot shows a web browser displaying the SENAI Editora website at [senaispeditora.com.br](https://www.senaispeditora.com.br). The page is titled 'destaques' (Highlights) and lists five books:

Imagem do Livro	Título do Livro	Preço
	design e economia circular	R\$59,90
	Guia Essencial para Gestão de Empresas	R\$120,00
	INSTALAÇÃO DE SISTEMA DE MICROGERAÇÃO SOLAR FOTOVOLTAICA	R\$78,00
	GESTÃO E EFICIÊNCIA ENERGÉTICA	R\$56,00
	COMUNICAÇÃO COM CORES	R\$58,00
	BLOCKCHAIN REVOLUTION	R\$98,00

Below the book list, there is a newsletter sign-up form with fields for Name and Email, and a 'cadastrar' button. There is also a link to activate Windows.

Fonte: <https://www.senaispeditora.com.br/>

Modificadores de acesso

Ao criarmos uma instância da classe **Produto**, não teremos acesso às propriedades e aos métodos da classe. Para que possamos ler e escrever nessas propriedades, podemos atribuir um modificador. Os modificadores de acesso são palavras-chave utilizadas para especificar a acessibilidade de uma propriedade, de um método ou de um acesso, como público (*public*), protegido (*protected*) ou privado (*private*).



Utilizando novamente o exemplo da **classe Produto**, delimitaremos o acesso ao nome como público e ao preço como **private**. Dessa maneira, você terá acesso somente ao nome, mas não ao preço, pois este estará limitado à própria classe.

The screenshot shows two open files in Visual Studio Code:

- Program.cs**:

```
1 using System;
2
3 namespace OOP
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9             Produto p = new Produto();
10            p.
11        }
12    }
13 }
```
- Produto.cs**:

```
1 namespace OOP
2 {
3     class Produto
4     {
5         public string nome; // propriedade nome
6         private float preco; // propriedade preco
7         public void CalcularDesconto() { } // métodos
8     }
9 }
```

A tooltip is displayed over the variable `p` in the `Program.cs` code editor, listing the following members of the `Produto` class:

- `CalcularDesconto`
- `Equals`
- `GetHashCode`
- `GetType`
- `nome`
- `ToString`

Uma das maneiras de se criar listas de objetos no C# é utilizando a classe **List<T>**, em que T é o tipo do item que a coleção armazenará.

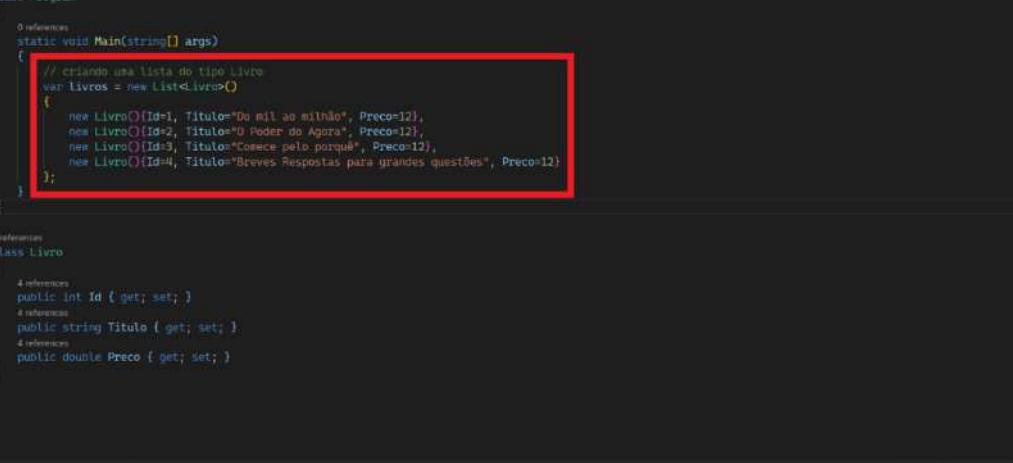
A seguir, conheça a construção do código para a classe **Livro**.

Codificação Back-end - SA2 - Desafio 2

The screenshot shows the Visual Studio Code interface. The left sidebar contains icons for file operations like Open, Save, Find, and Run. The main area displays the following C# code:

```
File Edit Selection View Go Run Terminal Help  
Program.cs - OOP - Visual Studio Code [Administrator]  
Program.cs > {} OOP > OOP.Program > Main(string[] args)  
20  
21     5 references  
22     class Livro  
23     {  
24         4 references  
25             public int Id { get; set; }  
26         4 references  
27             public string Titulo { get; set; }  
28         4 references  
29             public double Preco { get; set; }  
30     }  
31  
32 }  
33  
34  
35  
36  
37  
38  
39  
39 }  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
49 }  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
59 }  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
69 }  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
79 }  
80  
81  
82  
83  
84  
85  
86  
87  
88  
88 }  
89  
89 }  
90  
91  
92  
93  
94  
95  
95 }  
96  
97  
98  
99  
99 }  
100  
101  
102  
103  
104  
105  
105 }  
106  
107  
108  
109  
109 }  
110  
111  
112  
113  
113 }  
114  
115  
116  
117  
117 }  
118  
119  
119 }  
120  
121  
122  
123  
123 }  
124  
125  
125 }  
126  
127  
127 }  
128  
129  
129 }  
130  
131  
132  
133  
133 }  
134  
135  
135 }  
136  
137  
137 }  
138  
139  
139 }  
140  
141  
141 }  
142  
143  
143 }  
144  
145  
145 }  
146  
147  
147 }  
148  
149  
149 }  
150  
151  
151 }  
152  
153  
153 }  
154  
155  
155 }  
156  
157  
157 }  
158  
159  
159 }  
160  
161  
161 }  
162  
163  
163 }  
164  
165  
165 }  
166  
167  
167 }  
168  
169  
169 }  
170  
171  
171 }  
172  
173  
173 }  
174  
175  
175 }  
176  
177  
177 }  
178  
179  
179 }  
180  
181  
181 }  
182  
183  
183 }  
184  
185  
185 }  
186  
187  
187 }  
188  
189  
189 }  
190  
191  
191 }  
192  
193  
193 }  
194  
195  
195 }  
196  
197  
197 }  
198  
199  
199 }  
200  
201  
201 }  
202  
203  
203 }  
204  
205  
205 }  
206  
207  
207 }  
208  
209  
209 }  
210  
211  
211 }  
212  
213  
213 }  
214  
215  
215 }  
216  
217  
217 }  
218  
219  
219 }  
220  
221  
221 }  
222  
223  
223 }  
224  
225  
225 }  
226  
227  
227 }  
228  
229  
229 }  
230  
231  
231 }  
232  
233  
233 }  
234  
235  
235 }  
236  
237  
237 }  
238  
239  
239 }  
240  
241  
241 }  
242  
243  
243 }  
244  
245  
245 }  
246  
247  
247 }  
248  
249  
249 }  
250  
251  
251 }  
252  
253  
253 }  
254  
255  
255 }  
256  
257  
257 }  
258  
259  
259 }  
260  
261  
261 }  
262  
263  
263 }  
264  
265  
265 }  
266  
267  
267 }  
268  
269  
269 }  
270  
271  
271 }  
272  
273  
273 }  
274  
275  
275 }  
276  
277  
277 }  
278  
279  
279 }  
280  
281  
281 }  
282  
283  
283 }  
284  
285  
285 }  
286  
287  
287 }  
288  
289  
289 }  
290  
291  
291 }  
292  
293  
293 }  
294  
295  
295 }  
296  
297  
297 }  
298  
299  
299 }  
300  
301  
301 }  
302  
303  
303 }  
304  
305  
305 }  
306  
307  
307 }  
308  
309  
309 }  
310  
311  
311 }  
312  
313  
313 }  
314  
315  
315 }  
316  
317  
317 }  
318  
319  
319 }  
320  
321  
321 }  
322  
323  
323 }  
324  
325  
325 }  
326  
327  
327 }  
328  
329  
329 }  
330  
331  
331 }  
332  
333  
333 }  
334  
335  
335 }  
336  
337  
337 }  
338  
339  
339 }  
340  
341  
341 }  
342  
343  
343 }  
344  
345  
345 }  
346  
347  
347 }  
348  
349  
349 }  
350  
351  
351 }  
352  
353  
353 }  
354  
355  
355 }  
356  
357  
357 }  
358  
359  
359 }  
360  
361  
361 }  
362  
363  
363 }  
364  
365  
365 }  
366  
367  
367 }  
368  
369  
369 }  
370  
371  
371 }  
372  
373  
373 }  
374  
375  
375 }  
376  
377  
377 }  
378  
379  
379 }  
380  
381  
381 }  
382  
383  
383 }  
384  
385  
385 }  
386  
387  
387 }  
388  
389  
389 }  
390  
391  
391 }  
392  
393  
393 }  
394  
395  
395 }  
396  
397  
397 }  
398  
399  
399 }  
400  
401  
401 }  
402  
403  
403 }  
404  
405  
405 }  
406  
407  
407 }  
408  
409  
409 }  
410  
411  
411 }  
412  
413  
413 }  
414  
415  
415 }  
416  
417  
417 }  
418  
419  
419 }  
420  
421  
421 }  
422  
423  
423 }  
424  
425  
425 }  
426  
427  
427 }  
428  
429  
429 }  
430  
431  
431 }  
432  
433  
433 }  
434  
435  
435 }  
436  
437  
437 }  
438  
439  
439 }  
440  
441  
441 }  
442  
443  
443 }  
444  
445  
445 }  
446  
447  
447 }  
448  
449  
449 }  
450  
451  
451 }  
452  
453  
453 }  
454  
455  
455 }  
456  
457  
457 }  
458  
459  
459 }  
460  
461  
461 }  
462  
463  
463 }  
464  
465  
465 }  
466  
467  
467 }  
468  
469  
469 }  
470  
471  
471 }  
472  
473  
473 }  
474  
475  
475 }  
476  
477  
477 }  
478  
479  
479 }  
480  
481  
481 }  
482  
483  
483 }  
484  
485  
485 }  
486  
487  
487 }  
488  
489  
489 }  
490  
491  
491 }  
492  
493  
493 }  
494  
495  
495 }  
496  
497  
497 }  
498  
499  
499 }  
500  
501  
501 }  
502  
503  
503 }  
504  
505  
505 }  
506  
507  
507 }  
508  
509  
509 }  
510  
511  
511 }  
512  
513  
513 }  
514  
515  
515 }  
516  
517  
517 }  
518  
519  
519 }  
520  
521  
521 }  
522  
523  
523 }  
524  
525  
525 }  
526  
527  
527 }  
528  
529  
529 }  
530  
531  
531 }  
532  
533  
533 }  
534  
535  
535 }  
536  
537  
537 }  
538  
539  
539 }  
540  
541  
541 }  
542  
543  
543 }  
544  
545  
545 }  
546  
547  
547 }  
548  
549  
549 }  
550  
551  
551 }  
552  
553  
553 }  
554  
555  
555 }  
556  
557  
557 }  
558  
559  
559 }  
5510  
5511  
5511 }  
5512  
5513  
5513 }  
5514  
5515  
5515 }  
5516  
5517  
5517 }  
5518  
5519  
5519 }  
5520  
5521  
5521 }  
5522  
5523  
5523 }  
5524  
5525  
5525 }  
5526  
5527  
5527 }  
5528  
5529  
5529 }  
5530  
5531  
5531 }  
5532  
5533  
5533 }  
5534  
5535  
5535 }  
5536  
5537  
5537 }  
5538  
5539  
5539 }  
5540  
5541  
5541 }  
5542  
5543  
5543 }  
5544  
5545  
5545 }  
5546  
5547  
5547 }  
5548  
5549  
5549 }  
5550  
5551  
5551 }  
5552  
5553  
5553 }  
5554  
5555  
5555 }  
5556  
5557  
5557 }  
5558  
5559  
5559 }  
5560  
5561  
5561 }  
5562  
5563  
5563 }  
5564  
5565  
5565 }  
5566  
5567  
5567 }  
5568  
5569  
5569 }  
5570  
5571  
5571 }  
5572  
5573  
5573 }  
5574  
5575  
5575 }  
5576  
5577  
5577 }  
5578  
5579  
5579 }  
5580  
5581  
5581 }  
5582  
5583  
5583 }  
5584  
5585  
5585 }  
5586  
5587  
5587 }  
5588  
5589  
5589 }  
5590  
5591  
5591 }  
5592  
5593  
5593 }  
5594  
5595  
5595 }  
5596  
5597  
5597 }  
5598  
5599  
5599 }  
55100  
55101  
55101 }  
55102  
55103  
55103 }  
55104  
55105  
55105 }  
55106  
55107  
55107 }  
55108  
55109  
55109 }  
55110  
55111  
55111 }  
55112  
55113  
55113 }  
55114  
55115  
55115 }  
55116  
55117  
55117 }  
55118  
55119  
55119 }  
55120  
55121  
55121 }  
55122  
55123  
55123 }  
55124  
55125  
55125 }  
55126  
55127  
55127 }  
55128  
55129  
55129 }  
55130  
55131  
55131 }  
55132  
55133  
55133 }  
55134  
55135  
55135 }  
55136  
55137  
55137 }  
55138  
55139  
55139 }  
55140  
55141  
55141 }  
55142  
55143  
55143 }  
55144  
55145  
55145 }  
55146  
55147  
55147 }  
55148  
55149  
55149 }  
55150  
55151  
55151 }  
55152  
55153  
55153 }  
55154  
55155  
55155 }  
55156  
55157  
55157 }  
55158  
55159  
55159 }  
55160  
55161  
55161 }  
55162  
55163  
55163 }  
55164  
55165  
55165 }  
55166  
55167  
55167 }  
55168  
55169  
55169 }  
55170  
55171  
55171 }  
55172  
55173  
55173 }  
55174  
55175  
55175 }  
55176  
55177  
55177 }  
55178  
55179  
55179 }  
55180  
55181  
55181 }  
55182  
55183  
55183 }  
55184  
55185  
55185 }  
55186  
55187  
55187 }  
55188  
55189  
55189 }  
55190  
55191  
55191 }  
55192  
55193  
55193 }  
55194  
55195  
55195 }  
55196  
55197  
55197 }  
55198  
55199  
55199 }  
55200  
55201  
55201 }  
55202  
55203  
55203 }  
55204  
55205  
55205 }  
55206  
55207  
55207 }  
55208  
55209  
55209 }  
55210  
55211  
55211 }  
55212  
55213  
55213 }  
55214  
55215  
55215 }  
55216  
55217  
55217 }  
55218  
55219  
55219 }  
55220  
55221  
55221 }  
55222  
55223  
55223 }  
55224  
55225  
55225 }  
55226  
55227  
55227 }  
55228  
55229  
55229 }  
55230  
55231  
55231 }  
55232  
55233  
55233 }  
55234  
55235  
55235 }  
55236  
55237  
55237 }  
55238  
55239  
55239 }  
55240  
55241  
55241 }  
55242  
55243  
55243 }  
55244  
55245  
55245 }  
55246  
55247  
55247 }  
55248  
55249  
55249 }  
55250  
55251  
55251 }  
55252  
55253  
55253 }  
55254  
55255  
55255 }  
55256  
55257  
55257 }  
55258  
55259  
55259 }  
55260  
55261  
55261 }  
55262  
55263  
55263 }  
55264  
55265  
55265 }  
55266  
55267  
55267 }  
55268  
55269  
55269 }  
55270  
55271  
55271 }  
55272  
55273  
55273 }  
55274  
55275  
55275 }  
55276  
55277  
55277 }  
55278  
55279  
55279 }  
55280  
55281  
55281 }  
55282  
55283  
55283 }  
55284  
55285  
55285 }  
55286  
55287  
55287 }  
55288  
55289  
55289 }  
55290  
55291  
55291 }  
55292  
55293  
55293 }  
55294  
55295  
55295 }  
55296  
55297  
55297 }  
55298  
55299  
55299 }  
552100  
552101  
552101 }  
552102  
552103  
552103 }  
552104  
552105  
552105 }  
552106  
552107  
552107 }  
552108  
552109  
552109 }  
552110  
552111  
552111 }  
552112  
552113  
552113 }  
552114  
552115  
552115 }  
552116  
552117  
552117 }  
552118  
552119  
552119 }  
552120  
552121  
552121 }  
552122  
552123  
552123 }  
552124  
552125  
552125 }  
552126  
552127  
552127 }  
552128  
552129  
552129 }  
552130  
552131  
552131 }  
552132  
552133  
552133 }  
552134  
552135  
552135 }  
552136  
552137  
552137 }  
552138  
552139  
552139 }  
552140  
552141  
552141 }  
552142  
552143  
552143 }  
552144  
552145  
552145 }  
552146  
552147  
552147 }  
552148  
552149  
552149 }  
552150  
552151  
552151 }  
552152  
552153  
552153 }  
552154  
552155  
552155 }  
552156  
552157  
552157 }  
552158  
552159  
552159 }  
552160  
552161  
552161 }  
552162  
552163  
552163 }  
552164  
552165  
552165 }  
552166  
552167  
552167 }  
552168  
552169  
552169 }  
552170  
552171  
552171 }  
552172  
552173  
552173 }  
552174  
552175  
552175 }  
552176  
552177  
552177 }  
552178  
552179  
552179 }  
552180  
552181  
552181 }  
552182  
552183  
552183 }  
552184  
552185  
552185 }  
552186  
552187  
552187 }  
552188  
552189  
552189 }  
552190  
552191  
552191 }  
552192  
552193  
552193 }  
552194  
552195  
552195 }  
552196  
552197  
552197 }  
552198  
552199  
552199 }  
552200  
552201  
552201 }  
552202  
552203  
552203 }  
552204  
552205  
552205 }  
552206  
552207  
552207 }  
552208  
552209  
552209 }  
552210  
552211  
552211 }  
552212  
552213  
552213 }  
552214  
552215  
552215 }  
552216  
552217  
552217 }  
552218  
552219  
552219 }  
552220  
552221  
552221 }  
552222  
552223  
552223 }  
552224  
552225  
552225 }  
552226  
552227  
552227 }  
552228  
552229  
552229 }  
552230  
552231  
552231 }  
552232  
552233  
552233 }  
552234  
552235  
552235 }  
552236  
552237  
552237 }  
552238  
552239  
552239 }  
552240  
552241  
552241 }  
552242  
552243  
552243 }  
552244  
552245  
552245 }  
552246  
552247  
552247 }  
552248  
552249  
552249 }  
552250  
552251  
552251 }  
552252  
552253  
552253 }  
552254  
552255  
552255 }  
552256  
552257  
552257 }  
552258  
552259  
552259 }  
552260  
552261  
552261 }  
552262  
552263  
552263 }  
552264  
552265  
552265 }  
552266  
552267  
552267 }  
552268  
552269  
552269 }  
552270  
552271  
552271 }  
552272  
552273  
552273 }  
552274  
552275  
552275 }  
552276  
552277  
552277 }  
552278  
552279  
552279 }  
552280  
552281  
552281 }  
552282  
552283  
552283 }  
552284  
552285  
552285 }  
552286  
552287  
552287 }  
552288  
552289  
552289 }  
552290  
552291  
552291 }  
552292  
552293  
552293 }  
552294  
552295  
552295 }  
552296  
552297  
552297 }  
552298  
552299  
552299 }  
5522100  
5522101  
5522101 }  
5522102  
5522103  
5522103 }  
5522104  
5522105  
5522105 }  
5522106  
5522107  
5522107 }  
5522108  
5522109  
5522109 }  
5522110  
5522111  
5522111 }  
5522112  
5522113  
5522113 }  
5522114  
5522115  
5522115 }  
5522116  
5522117  
5522117 }  
5522118  
5522119  
5522119 }  
5522120  
5522121  
5522121 }  
5522122  
5522123  
5522123 }  
5522124  
5522125  
5522125 }  
5522126  
5522127  
5522127 }  
5522128  
5522129  
5522129 }  
5522130  
5522131  
5522131 }  
5522132  
5522133  
5522133 }  
5522134  
5522135  
5522135 }  
5522136  
5522137  
5522137 }  
5522138  
5522139  
5522139 }  
5522140  
5522141  
5522141 }  
5522142  
5522143  
5522143 }  
5522144  
5522145  
5522145 }  
5522146  
5522147  
5522147 }  
5522148  
5522149  
5522149 }  
5522150  
5522151  
5522151 }  
5522152  
5522153  
5522153 }  
5522154  
5522155  
5522155 }  
5522156  
5522157  
5522157 }  
5522158  
5522159  
5522159 }  
5522160  
5522161  
5522161 }  
5522162  
5522163  
5522163 }  
5522164  
5522165  
5522165 }  
5522166  
5522167  
5522167 }  
5522168  
5522169  
5522169 }  
5522170  
5522171  
5522171 }  
5522172  
5522173  
5522173 }  
5522174  
5522175  
5522175 }  
5522176  
5522177  
5522177 }  
5522178  
5522179  
5522179 }  
5522180  
5522181  
5522181 }  
5522182  
5522183  
5522183 }  
5522184  
5522185  
5522185 }  
5522186  
5522187  
5522187 }  
5522188  
5522189  
5522189 }  
5522190  
5522191  
5522191 }  
5522192  
5522193  
5522193 }  
5522194  
5522195  
5522195 }  
5522196  
5522197  
5522197 }  
5522198  
5522199  
5522199 }  
5522200  
5522201  
5522201 }  
5522202  
5522203  
5522203 }  
5522204  
5522205  
5522205 }  
5522206  
5522207  
5522207 }  
5522208  
5522209  
5522209 }  
5522210  
5522211  
5522211 }  
5522212  
5522213  
5522213 }  
5522214  
5522215  
5522215 }  
5522216  
5522217  
5522217 }  
5522218  
5522219  
5522219 }  
5522220  
5522221  
5522221 }  
5522222  
5522223  
5522223 }  
5522224  
5522225  
5522225 }  
5522226  
5522227  
5522227 }  
5522228  
5522229  
5522229 }  
5522230  
5522231  
5522231 }  
5522232  
5522233  
5522233 }  
5522234  
5522235  
5522235 }  
5522236  
5522237  
5522237 }  
5522238  
5522239  
5522239 }  
5522240  
5522241  
5522241 }  
5522242  
5522243  
5522243 }  
5522244  
5522245  
5522245 }  
5522246  
5522247  
552
```

No destaque, temos um exemplo de construção de código para a criação de uma lista de objetos do tipo `livro`.

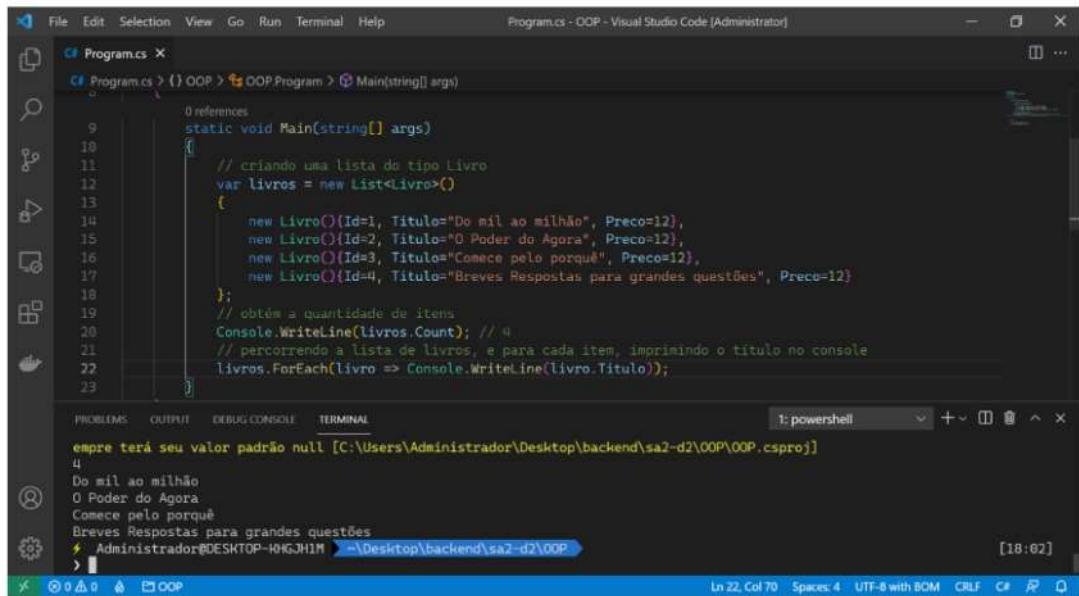


```
File Edit Selection View Go Run Terminal Help
Programs - OOP - Visual Studio Code [Administrator]
Program.cs > {} OOP > OOP.Program

7 0 references
8 class Program
9
10 static void Main(string[] args)
11 {
12     // criando uma lista do tipo Livro
13     var livros = new List<Livro>()
14     {
15         new Livro(){Id=1, Titulo="Do mil ao milhão", Preco=12},
16         new Livro(){Id=2, Titulo="O Poder do Agora", Preco=12},
17         new Livro(){Id=3, Titulo="Cosece pelo porquê", Preco=12},
18         new Livro(){Id=4, Titulo="Breves Respostas para grandes questões", Preco=12}
19     };
20 }
21
22 5 references
23 class Livro
24 {
25     4 references
26     public int Id { get; set; }
27     4 references
28     public string Titulo { get; set; }
29     4 references
30     public double Preco { get; set; }
}

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Administrator@DESKTOP-IHGJHM ~Desktop\backend\sistemaOOP
[17:41] 1: powershell + + + + x
[17:41]
```

A classe `List` fornece métodos para manipular a lista. Exemplos:



The screenshot shows a Visual Studio Code window with the following code in Program.cs:

```
File Edit Selection View Go Run Terminal Help
Program.cs - OOP - Visual Studio Code [Administrator]
Program.cs > {} OOP > OOP.Program > Main(string[] args)
0 referências
static void Main(string[] args)
{
    // criando uma lista do tipo Livro
    var livros = new List<Livro>()
    {
        new Livro(){Id=1, Titulo="Do mil ao milhão", Preco=12},
        new Livro(){Id=2, Titulo="O Poder do Agora", Preco=12},
        new Livro(){Id=3, Titulo="Comece pelo porquê", Preco=12},
        new Livro(){Id=4, Titulo="Breves Respostas para grandes questões", Preco=12}
    };
    // obtém a quantidade de itens
    Console.WriteLine(livros.Count); // 4
    // percorrendo a lista de livros, e para cada item, imprimindo o título no console
    livros.ForEach(livro => Console.WriteLine(livro.Titulo));
}

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
empre terá seu valor padrão null [C:\Users\Administrador\Desktop\backend\sa2-d2\OOP\OOP.csproj]
4
Do mil ao milhão
O Poder do Agora
Comece pelo porquê
Breves Respostas para grandes questões
Administrator@DESKTOP-KHGJHIM ~\Desktop\backend\sa2-d2\OOP
[18:02]
Ln 22, Col 70 Spaces: 4 UTF-8 with BOM CRLF CR LF Q
```

Saiba mais

Acesse o link: <https://docs.microsoft.com/pt-br/dotnet/api/system.collections.generic.list-1?view=net-5.0> para mais informações sobre como trabalhar com listas.



CRIANDO A MODELAGEM ORIENTADA A OBJETOS

Quando desenvolvemos uma aplicação, é importante entender como o sistema irá se comunicar por completo, ou seja, o usuário se comunica com o sistema, e este realiza a comunicação internamente. Por exemplo, o que o sistema irá fazer quando o usuário informar um dado? Salvará a informação no banco de dados, ou seja, usuário para sistema, depois, sistema para banco de dados.

A modelagem tem por objetivo fazer com que desenvolvedores entendam o sistema por completo. Trata-se da construção de modelos que explicam ou ilustram a estrutura ou organização do sistema, que pode ser um software, um site (entre outros) ou sua dinâmica de funcionamento. Há várias técnicas e ferramentas de modelagem existentes e cada uma delas é adequada a um determinado problema.

A Linguagem de Modelagem Unificada (UML) é uma linguagem-padrão que auxilia na modelagem sistemas. Esta linguagem é expressa através de diagramas. Cada

diagrama é composto por elementos (formas gráficas usadas para os desenhos) que possuem relação entre si.

Um dos tipos mais populares é o diagrama de classes, que pode ser utilizado como base para a criação de uma solução orientada a objetos, pois mapeiam de forma clara a estrutura de um determinado sistema ao modelar suas classes, seus atributos, operações e relações entre objetos.

No diagrama de classes, você pode representar as classes da seguinte maneira:



O nome da classe é o primeiro dado a ser inserido; os atributos são posicionados na parte intermediária e podem ser classificados como – (privados) ou + (públicos). Por fim, na parte inferior, são inseridos os métodos da classe.

Os relacionamentos entre as classes são representados pelas formas: herança, associação, agregação e composição, conforme exemplo a seguir.



Saiba mais

Acesse o link: <https://www.lucidchart.com/pages/pt/modelos-e-exemplos-de-diagramas-uml> e conheça exemplos de diagramas de classe.



Existem algumas ferramentas gratuitas para criar diagramas, como o **Draw.io**, um editor gráfico que possui uma área exclusiva para itens da UML.

CONCEITOS DE OO

Até aqui, você conheceu os conceitos de **classe**, **objeto** e modelagem que integram a POO. Agora, você vai estudar como abstrair detalhes de implementação e como reutilizar um código.

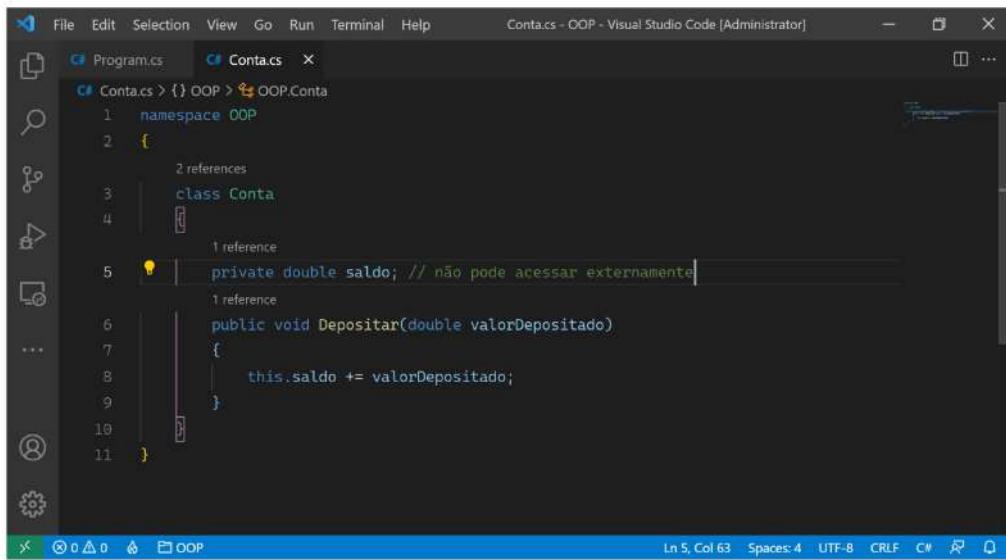
Encapsulamento

Encapsulamento (algumas vezes chamado de ocultamento de dados) é um conceito importante para o trabalho com objetos. Trata-se de reunir ("encapsular") as propriedades e os métodos de uma determinada classe ou objeto, tornando o desenvolvimento e a manutenção do software mais simples, protegendo os dados de modificações e escondendo os detalhes internos da implementação. Cada objeto deve controlar e manipular o seu próprio estado.

O encapsulamento é um dos principais conceitos da orientação a objetos e seu entendimento é simples: trata-se de combinar propriedades e métodos em um pacote e ocultar a sua implementação (propriedades e métodos). **Não importa como a classe faz o trabalho, mas sim o que ela faz.** Utilizar a visibilidade das propriedades e dos métodos (por exemplo, public ou private) é uma das maneiras de ocultar ou deixar acessíveis informações.

Por exemplo, a quantidade de dinheiro disponível em uma conta bancária somente pode ser movimentada por meio de transações bancárias. Não há como acessar o dinheiro diretamente nos cofres do banco.

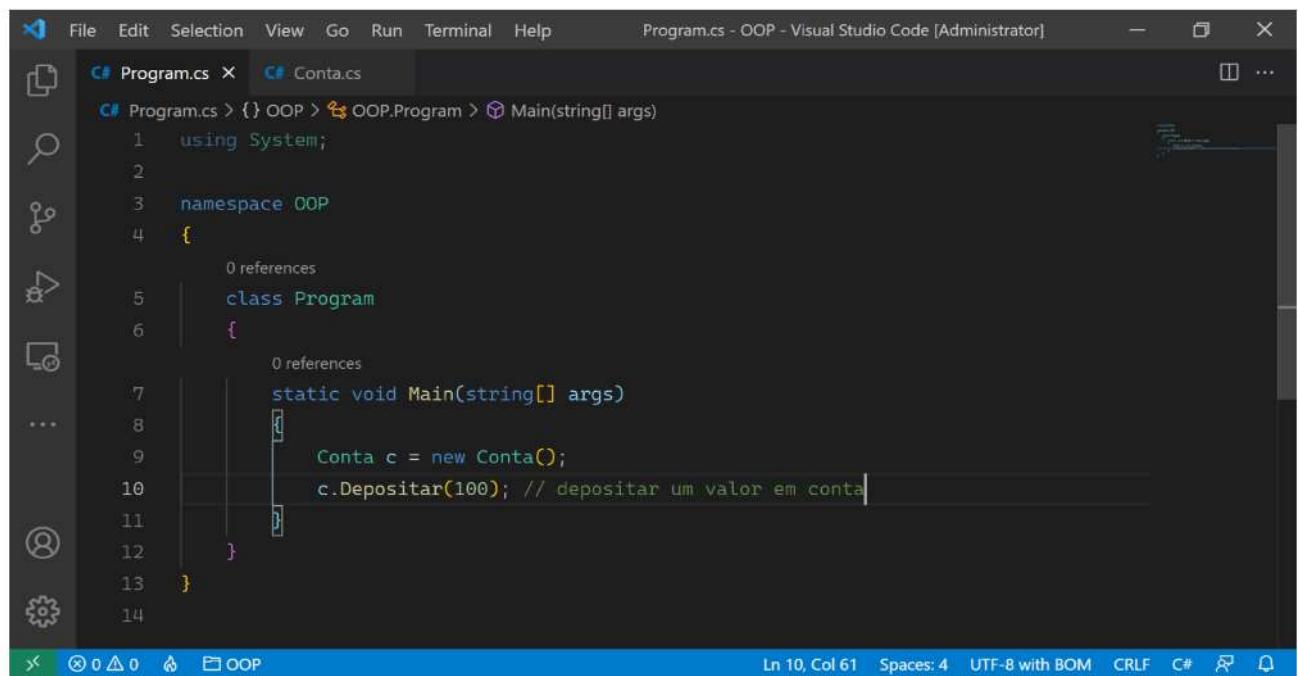
A classe **Conta** possui um saldo que não pode ser acessado externamente. Nesse caso, o que você pode fazer é manipulá-lo, por meio do método: **Depositar()**.



```
File Edit Selection View Go Run Terminal Help
Contas.cs - OOP - Visual Studio Code [Administrator]
Program.cs Conta.cs
Contas.cs > {} OOP > OOP.Conta
1 namespace OOP
2 {
3     2 references
4     class Conta
5     {
6         1 reference
7         private double saldo; // não pode acessar externamente
8         1 reference
9         public void Depositar(double valorDepositado)
10        {
11            this.saldo += valorDepositado;
12        }
13    }
14}

Ln 5, Col 63 Spaces: 4 UTF-8 CRLF C# R Q
```

A seguir, temos um exemplo no qual o método da classe **Conta Depositar()** é chamado no Program.cs.



```
File Edit Selection View Go Run Terminal Help
Program.cs - OOP - Visual Studio Code [Administrator]
Program.cs Conta.cs
Program.cs > {} OOP > OOP.Program > Main(string[] args)
1 using System;
2
3 namespace OOP
4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main(string[] args)
10        {
11            Conta c = new Conta();
12            c.Depositar(100); // depositar um valor em conta
13        }
14    }
15}

Ln 10, Col 61 Spaces: 4 UTF-8 with BOM CRLF C# R Q
```

Properties

Um problema possível para o cenário apresentado no exemplo anterior é o bloqueio de escrita e de leitura da propriedade determinado pelo modificador **private**. Para casos como esse, o C# oferece um recurso chamado properties.

O recurso properties permite que a classe disponibilize os valores de **{get}** e **{set}** escondendo à implementação ou verificação do código. Nesse caso, não queremos permitir que um saldo seja alterado diretamente, mas que ele possa ser consultado.

Na versão completa, temos a forma apresentada a seguir:

```
1. private double saldo
2. {
3.     get
4.     {
5.         //código para ler a propriedade
6.     }
7.     set
8.     {
9.         //código para escrever na propriedade
10.    }
11. }
```

Na versão simplificada:

```
1. private double saldo {get; set;}
```

Para tornar o saldo público e acessível por qualquer classe, devemos mudar a visibilidade da propriedade e manter a alteração do valor do saldo somente para a própria classe.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Conta.cs - OOP - Visual Studio Code [Administrator].
- Left Sidebar:** Includes icons for file operations like Open, Save, Find, and others.
- Code Editor:** Displays the following C# code:

```
1  namespace OOP
2  {
3      class Conta
4      {
5          public double saldo { get; private set; }
6
7          public void Depositar(double valorDepositado)
8          {
9              this.saldo += valorDepositado;
10         }
11     }
12 }
```
- Bottom Status Bar:** Ln 4, Col 6, Spaces: 4, UTF-8, CRLF, C#, etc.

No Program.cs, você poderá depositar um valor e consultar o saldo.

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** Program.cs - OOP - Visual Studio Code [Administrator].
- Left Sidebar:** Includes icons for file operations like Open, Save, Find, and others.
- Code Editor:** Displays the following C# code:

```
7  static void Main(string[] args)
8  {
9      Conta c = new Conta();
10     c.Depositar(100); // depositar um valor em conta
11     Console.WriteLine(c.saldo);
12 }
```
- Bottom Status Bar:** Ln 11, Col 40, Spaces: 4, UTF-8 with BOM, CRLF, C#, etc.
- Terminal:** Shows the command line output:

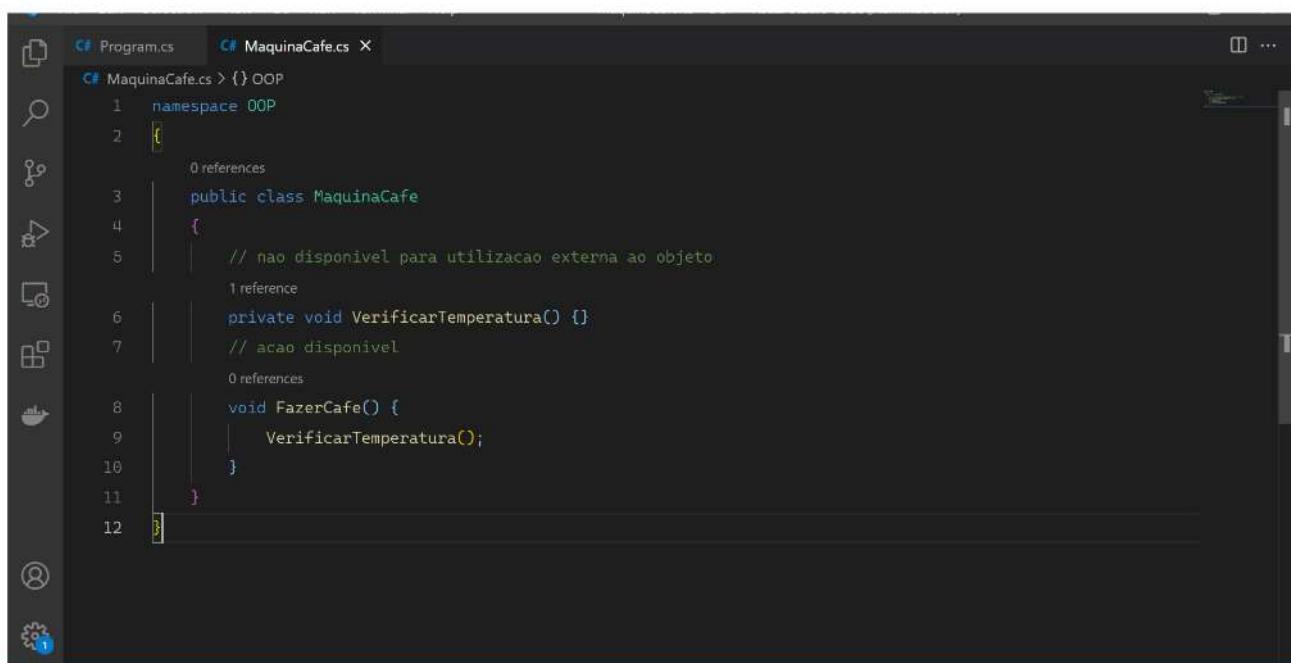
```
⚡ Administrador@DESKTOP-KHGJH1M ~\Desktop\backend\sa2-d2\OOP [12:47]
> dotnet run
100
⚡ Administrador@DESKTOP-KHGJH1M ~\Desktop\backend\sa2-d2\OOP [12:47]
> ⚡
```

Por convenção, ao utilizarmos as properties, escrevemos a primeira letra das palavras em maiúscula (o que também conhecido como PascalCase).

Abstração

No paradigma orientado a objetos, o conceito de abstração é mover o foco dos detalhes da implementação para as operações disponíveis, escondendo-os de um objeto externo.

Por exemplo, para utilizar uma máquina de café, você não precisa conhecer a temperatura ideal em que ela opera; você apenas precisa conhecer quais métodos estão disponíveis para acionar uma tarefa específica (adicionar açúcar ao café, por exemplo). Ou seja, você não precisa conhecer sobre como o método é implementado para criar o resultado esperado.



```
namespace OOP
{
    public class MaquinaCafe
    {
        // nao disponivel para utilizacao externa ao objeto
        private void VerificarTemperatura() {}

        // acao disponivel
        void FazerCafe()
        {
            VerificarTemperatura();
        }
    }
}
```

Relações entre classes

As relações mais comuns entre classes são:

Inclusão (“tem um”)

Herança (“é um”)

A relação de inclusão ocorre quando uma classe **Pedido** contém um **Cliente**, ou seja, quando objetos da classe A contêm objetos da classe B.

```
File Edit Selection View Go Run Terminal Help Pedido.cs - OOP - Visual Studio Code [Administrator]
C# Pedido.cs ×
C# Pedido.cs > {} OOP > OOP.Pedido > identificador
1 namespace OOP
2 {
3     class Pedido
4     {
5         int identificador;
6         Cliente cliente;
7     }
8 }

C# Cliente.cs ×
C# Cliente.cs > {} OOP > OOP.Cliente
1 namespace OOP
2 {
3     class Cliente
4     {
5         string nome;
6     }
7 }
```

A relação de herança, por sua vez, denota especialização. Por exemplo, as classes **ContaCorrente** e **ContaPoupança** serão herdeiras da classe **Conta**.

Vamos detalhar um pouco mais a relação de herança a seguir.

Herança

Consideramos a herança como uma maneira de reutilização de software, na qual uma nova classe é criada absorvendo as propriedades e os métodos da classe herdada, adicionando propriedades e métodos novos e eliminando a necessidade de reescrita do código.

PDF

Acesse seu material complementar e conheça um exemplo prático de herança.

Arquivo: 01_heranca.pdf

Polimorfismo

O termo “polimorfismo” é originário do grego e significa “muitas formas” (*poly* = muitas, *morphos* = formas). Por exemplo, no jogo de videogame Super Mario, há diferentes personagens que realizam, entre outras, uma mesma ação em comum: correr. O polimorfismo permite que cada personagem (Mario, Luigi e Yoshi) tenha seu próprio modo de correr.

PDF

Acesse seu material complementar e conheça um exemplo prático de polimorfismo.

Arquivo: 02_polimorfismo.pdf

Classes abstratas

São chamadas de classes abstratas as classes que servem somente como base para outras, ou seja, surgem quando não pretendemos criar objetos (instanciar) para essa classe. Assim, a classe pai, em algum ponto do código, torna-se tão geral que acaba sendo vista mais como um modelo para outras classes do que como uma classe com instâncias específicas para sua utilização.

PDF

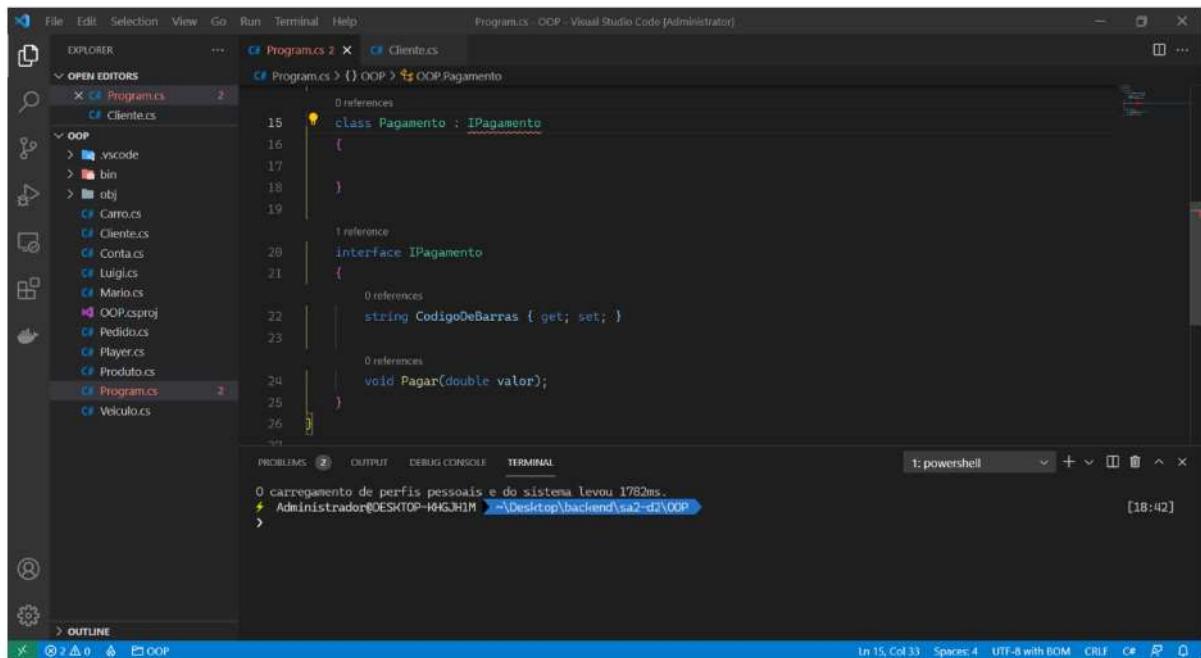
Acesse seu material complementar e conheça um exemplo prático de classes abstratas.

Arquivo: 03_classes_abstratas.pdf

Interface

Trata-se de um recurso utilizado na orientação a objetos para diminuir a dependência entre módulos do sistema (o que chamamos de **acoplamento**). A interface em programação é um tipo abstrato de dados que conterá somente a definição, mas não a implementação.

A implementação será feita na classe filha e é definida por meio da palavra reservada *interface*. Quando criada uma interface, cria-se um contrato que ambas as partes devem aceitar.



The screenshot shows the Visual Studio Code interface with the following details:

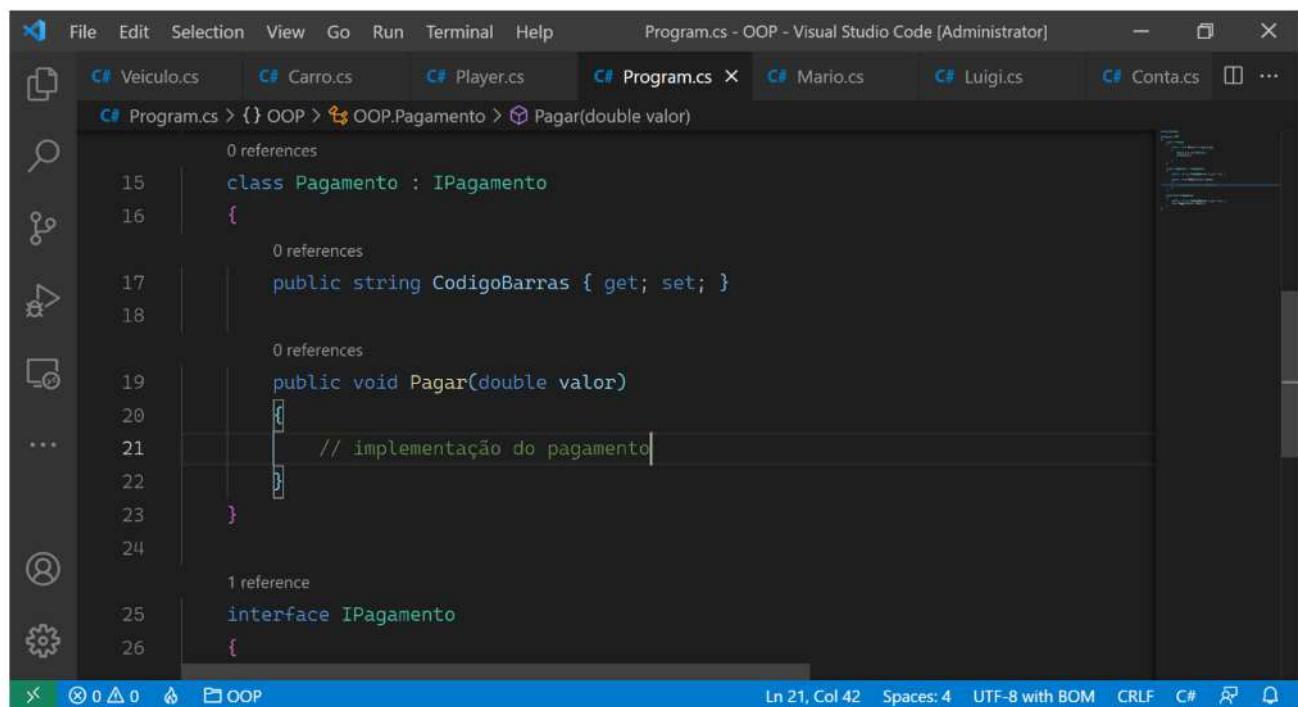
- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Area:** The file `Program.cs` is open, showing the implementation of the `IPagamento` interface. The code is as follows:

```
class Pagamento : IPagamento
{
}

interface IPagamento
{
    string CódigoDeBarras { get; set; }

    void Pagar(double valor);
}
```
- Explorer:** Shows the project structure under the `OOP` folder, including files like `Carro.cs`, `Cliente.cs`, `Conta.cs`, `Luigi.cs`, `Mario.cs`, `Pedido.cs`, `Player.cs`, `Produto.cs`, `Veiculo.cs`, and `Program.cs`.
- Terminal:** Shows the command line output: "0 carregamento de perfis pessoais e do sistema levou 1782ms." and the current directory: "Administrador@DESKTOP-KHGJHJM ~\Desktop\backend\sa2-d2\OOP". The time is listed as "[18:42]".
- Status Bar:** Shows the line number (Ln 15, Col 33), spaces (Spaces: 4), encoding (UTF-8 with BOM), and file type (C#).

Ao implementar a interface `IPagamento`, você deverá especificar (em `Pagamento`) como ela funcionará. Atente-se que, ao implementar uma interface, a classe filha obrigatoriamente deve conter a assinatura do método.



The screenshot shows the Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Editor Area:** The file `Program.cs` is open, showing the implementation of the `IPagamento` interface. The code is as follows:

```
class Pagamento : IPagamento
{
}

public string CódigoBarra { get; set; }

public void Pagar(double valor)
{
    // implementação do pagamento
}

interface IPagamento
{
}
```
- Explorer:** Shows the project structure under the `OOP` folder, including files like `Veiculo.cs`, `Carro.cs`, `Player.cs`, `Program.cs`, `Mario.cs`, `Luigi.cs`, and `Conta.cs`.
- Terminal:** Shows the command line output: "0 carregamento de perfis pessoais e do sistema levou 1782ms." and the current directory: "Administrador@DESKTOP-KHGJHJM ~\Desktop\backend\sa2-d2\OOP". The time is listed as "[18:42]".
- Status Bar:** Shows the line number (Ln 21, Col 42), spaces (Spaces: 4), encoding (UTF-8 with BOM), and file type (C#).

Saiba mais



Acesse o link: <https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop> e saiba mais sobre OOP (C#).

TRATAMENTO DE ERROS

Exceções podem ocorrer quando um programa estiver sendo executado, ou seja, situações atemporais e não esperadas pelo programa podem provocar um erro ou uma exceção. Para tratar essas situações excepcionais, utilizamos a instrução try-catch.

A sintaxe-base fundamental para o tratamento de exceções é:

```
1. try {  
2.     // bloco  
3. } catch (tipo exceção) {  
4.     // bloco  
5. }
```

```
File Edit Selection View Go Run Terminal Help Program.cs - OOP - Visual Studio Code [Administrator]  
Program.cs > {} OOP > OOP.Program > Main(string[] args)  
5 {  
    0 references  
    class Program  
    1 [  
        0 references  
    8 static void Main(string[] args)  
    9 {  
    10     11 try {  
    12         13     int[] numeros = { 1, 2, 3 };  
    14         15     Console.WriteLine(numeros[10]); // error!  
    16     }  
    17     18     catch (System.Exception ex)  
    19     {  
        20         21         Console.WriteLine("Vou cair aqui: " + ex.Message);  
    22     }  
    23 }  
    24 }  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
C:\Users\Administrador\Desktop\backend\sa2-d2\OOP\Pedido.cs(5,13): warning CS0169: O campo "Pedido.identificador" nunca é usado [C:\Users\Administrador\Desktop\backend\sa2-d2\OOP\OOP.csproj]  
Vou cair aqui: Index was outside the bounds of the array.  
Administrator@DESKTOP-IHGJH1M ~\Desktop\backend\sa2-d2\OOP  
1: powershell [21:24]  
In 18, Col 14 Spaces: 4 UTF-8 with BOM CRLF CR LF
```

DEPURAÇÃO

Ao depurar a aplicação, a ferramenta permite visualizar o que o código está fazendo enquanto está sendo executado, ou seja, você poderá observar os valores armazenados em uma variável, inspecionar os dados de uma classe e verificar os valores alterados durante o processo de execução. Isso permite ter uma melhor visualização e precisão da real execução do seu código com os valores esperados.

PDF

Acesse seu material complementar e conheça um exemplo prático de depuração.

Arquivo: 04_depuracao.pdf

NESTE DESAFIO...

Codificação Back-End | Desafio 2



Neste desafio, você estudou os conceitos relacionados à programação orientada a objetos, como paradigma, objeto, classe, interface, polimorfismo, abstração, encapsulamento e herança, que possibilitam a reutilização de código e a abstração de responsabilidades.

Além disso, você conheceu exemplos práticos desse tipo de programação, aplicando a linguagem C#.

No próximo desafio...

Você conhecerá como aplicar as boas práticas para elaborar a documentação de projetos.

DESAFIO 3

Nesta etapa, você deverá resolver o desafio 3:

- Utilizar linguagem de marcação para a apresentação dos recursos disponibilizados no problema.

Para isso, você estudará os seguintes conteúdos:

- Linguagem de marcação
- Estrutura de documentos
- Formatação
- Etiquetas para links
- Listas numeradas
- Listas não numeradas
- Tabelas
- Formulários



DOCUMENTAÇÃO

Documentação em engenharia de software é um termo que abrange todos os documentos escritos, imagens e materiais relacionados ao processo de desenvolvimento.

Documentações possuem diferentes propósitos e são utilizadas em diferentes contextos (baseados nos objetivos do projeto).

Saiba mais

Para saber mais sobre documentação, acesse os links:



<https://blog.prototypr.io/software-documentation-types-and-best-practices-1726ca595c7f> (material em inglês)

<https://ieeexplore.ieee.org/document/6408631> (material em inglês).

TIPOS DE DOCUMENTAÇÃO

Dependendo do objetivo, do público e da metodologia de desenvolvimento que você está trabalhando, você pode encontrar diferentes tipos de documentação.

Clique nas abas para conhecê-los.



Documentação de requisitos

Aborda requisitos funcionais e não funcionais. Requisitos funcionais são os objetivos que o projeto deve atender, como inserir dados através de um formulário, realizar compras, elaborar relatórios ou fazer a comunicação entre cliente e comerciante. Requisitos não funcionais é tudo que será necessário para atender ao objetivo, como tipo de sistema operacional, perfil de hardware necessário, banco de dados ou o tipo de dispositivo em que o software pode ser usado. Resumidamente, requisitos funcionais definem o que fazer e os requisitos não funcionais definem como chegar lá.

Documentação de arquitetura do sistema

Descreve os componentes do sistema. Pode conter o desenho técnico da solução, englobando o sistema de Back-End, Front-End, banco de dados e toda a arquitetura interna de cada componente.

Documentação técnica

Aborda a documentação de utilização da API, ou seja, o que a API faz e quais recursos oferece.

Documentação para usuário final

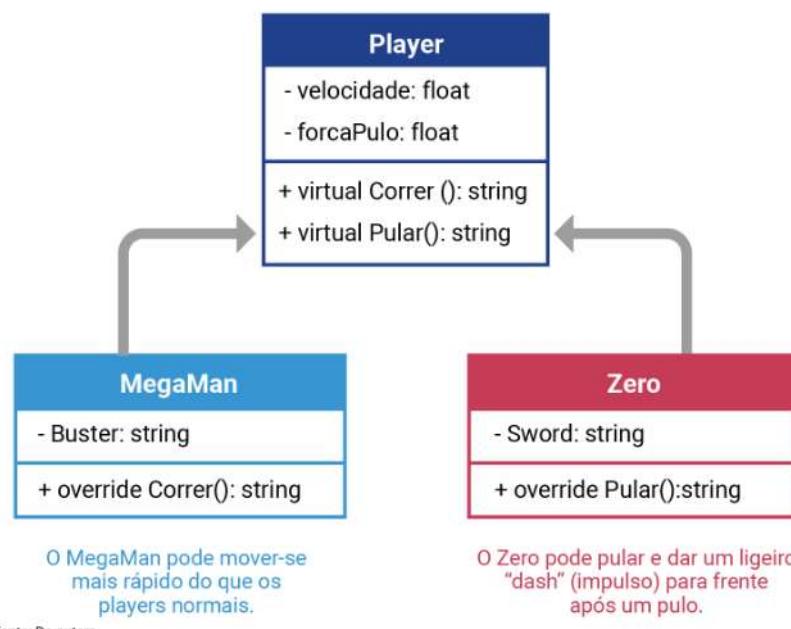
Aborda a utilização sobre como operar uma determinada aplicação, como instruções e manuais de usuário.

Documentação de suporte

Contém uma visão geral do sistema, tomadas de decisões e seu histórico. Também chamada de documentação geral de sistema.

A documentação por exemplo, pode conter o diagrama de classes, conforme mostrado no segundo desafio.

Nesse diagrama de classes, as classes MegaMan e Zero são classes herdeiras da classe Player, em que MegaMan e Zero possuem a função de correr, mas cada um deles corre de um jeito específico. A assinatura é a mesma, mas o comportamento é diferente em que chamamos de polimorfismo. Neste caso, o diagrama documenta com clareza as especificidades dessas classes.



Importante!



A decisão de criar a documentação não deve ser apenas uma obrigação. É importante pensar no porquê da validade do documento.

A documentação não interfere no funcionamento do código e, portanto, pode ser considerada opcional para a eficiência do projeto. Porém, de acordo com as boas práticas, a documentação do projeto é de extrema importância para o acompanhamento e aprimoramento do projeto.

Imagine que ruim comprar um item para montar sem um guia de instruções?

No mercado de trabalho, é comum encontrar tipos distintos de documentação:

- Histórico sobre a tomada de decisões;
- Validar um contrato entre partes diferentes da equipe, como, por exemplo, um contrato prévio do que será criado, modelado e validado por ambas as partes.
- Documentação técnica de escolha de ferramentas baseadas em configuração, instalação e pontos positivos da escolha da ferramenta.

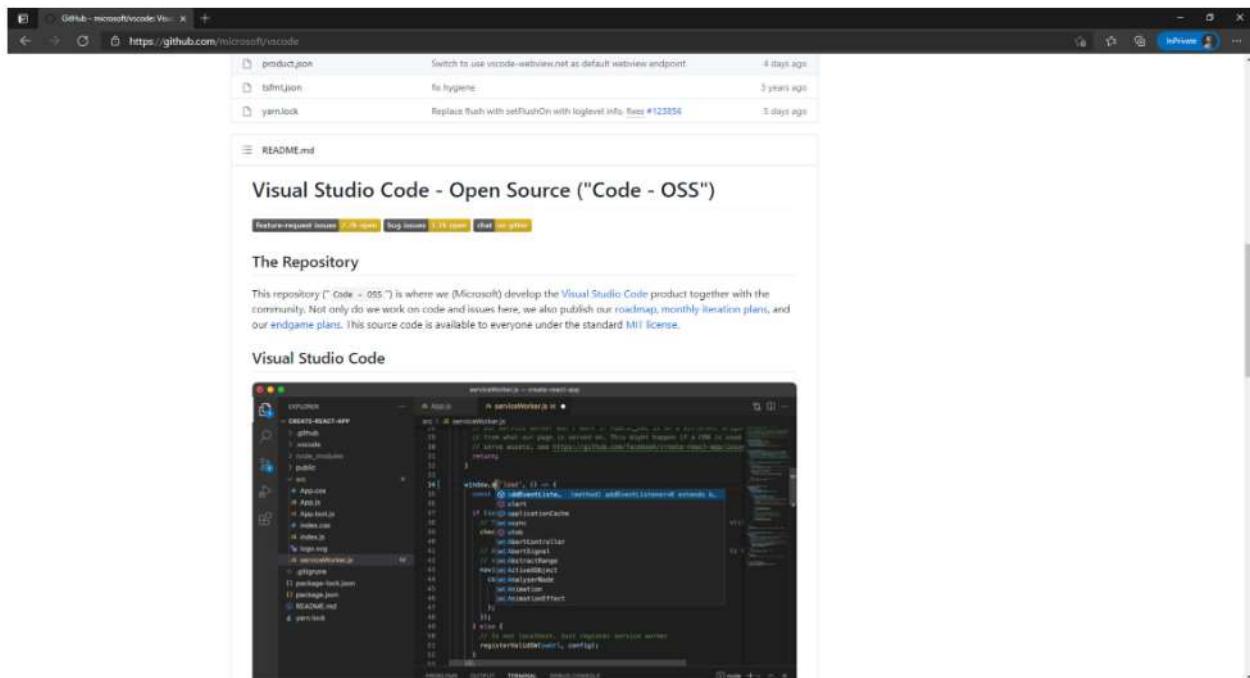
Outra situação comum em desenvolvimento de software é atuar mais em atualizações ou melhorias de projetos que já existem do que atuar em softwares desde o princípio. Por isso, a documentação do que foi implementado é importante.

PENSE NISSO...

Apenas pense na quantidade de aplicativos que você já consome por dia, os novos que você instala e na quantidade de sites que você acessa.

Aplicativos estão em constante aperfeiçoamento, surgindo novas versões frequentemente.

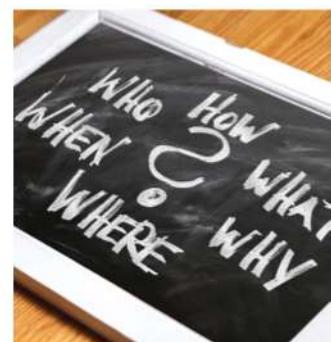
Com base nesse cenário, abordaremos uma documentação técnica, de desenvolvedor para desenvolvedor, chamada README.MD., conforme mostra o exemplo da imagem a seguir.



README.MD

README.md é um arquivo com extensão .md (Markdown), cujo objetivo é conter informações essenciais sobre o projeto.

Em outras palavras, o README.md é a documentação técnica geral do projeto.



O README.md é um arquivo utilizado para descrever, documentar ou exemplificar seu projeto. Apesar de não ser obrigatório (seu código vai funcionar perfeitamente sem ele), o README é essencial, pois mescla o cartão de visitas e a ementa do projeto, sendo o responsável por cativar interesse em seu trabalho ou relembrá-lo dos aspectos gerais e pontos relevantes daquele desenvolvimento.

Por boas práticas, o README.md sempre deve estar disponível, acessível e visível nos repositórios remotos.

MARKDOWN

Markdown Syntax é um tipo de linguagem de marcação, assim como o HTML. A sintaxe facilita a formatação de textos na web, funcionando como um conversor de texto para HTML.

O Markdown também utiliza tags, porém diferentes do HTML. Veja a lista de tags básicas para formatar seu README:

- títulos
- ênfase
- listas
- links
- tabelas
- imagens

PDF

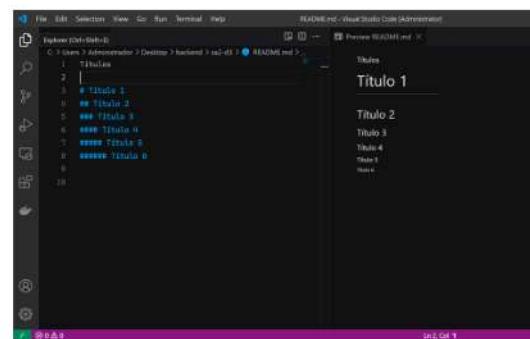
Acesse seu material complementar e acompanhe o passo a passo para criar um arquivo `readme.md` no VSCode.

Arquivo: `01_criando_readme.pdf`

conheça as principais tags para criar seu `readme`:

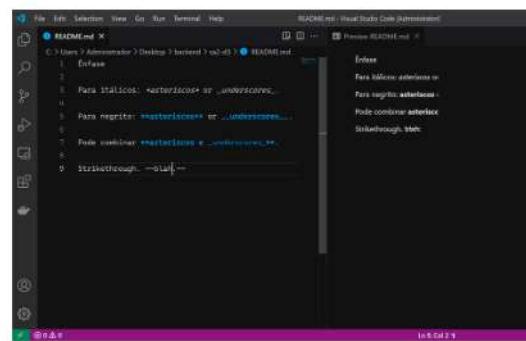
Títulos

A tag de título é a # (hashtag). Basta iniciar o título com a hashtag seguido do texto. Quanto maior o número de hashtags, menor o tamanho do título. Observe que o título 1 é automaticamente acompanhado de uma linha.



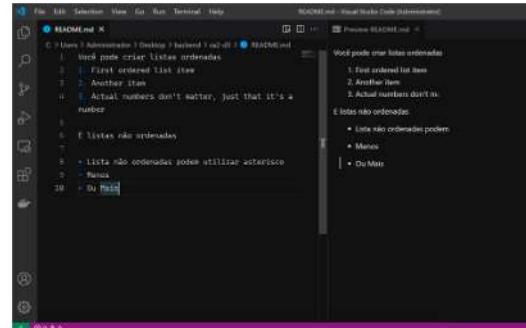
Ênfase

Para ênfase, usamos o texto entre asteriscos ou underscores para itálico e duplos asteriscos ou underscores para negrito. É possível combiná-los, como mostra a imagem.



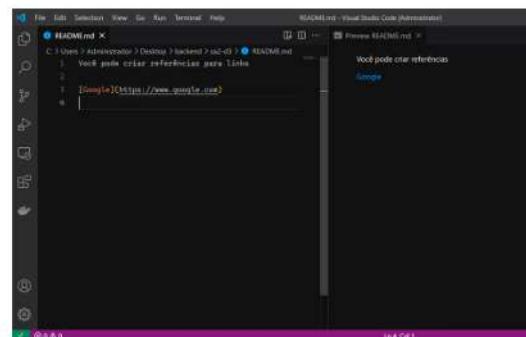
Listas

Para listas ordenadas, inicie o item com o número seguido de ponto. Para listas não-ordenadas, inicie a lista com um asterisco e cada item com o sinal de + ou -.



Link

Para criar um hipertexto, coloque o texto a ser clicado entre colchetes, seguido do endereço entre parênteses. O padrão é o link abrir em uma nova janela.



Tabela

Inicie a tabela com uma barra vertical ou pipe, que servem para dividir as colunas.

No destaque da imagem, os traços são linhas cheias. Os dois pontos marcam o alinhamento da coluna.

Tables	Ave	Grid
	right	1.000
col 2	centered	333
more	ave	\$1

Imagens

Para inserir uma imagem, use a tag , como no HTML.



Saiba mais

Há várias maneiras de formatar seu readme, pesquise! Para saber como inserir emojis, por exemplo, acesse o link: <https://github.com/ikatyang/emoji-cheat-sheet/blob/master/README.md>.



Para inserir uma formatação de fonte do tipo código de programação, acesse aqui: <https://docs.github.com/pt/github/writing-on-github/working-with-advanced-formatting/creating-and-highlighting-code-blocks>.

Para formatar tabelas, acesse: <https://docs.github.com/en/github/writing-on-github/working-with-advanced-formatting/organizing-information-with-tables>.

Para listas e outras dicas, acesse o link: <https://docs.github.com/en/github/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax#lists>.

CONSTRUINDO README

O foco do nosso readme será uma documentação mais técnica. O que seria necessário eu informar para outro colaborador para que ele possa ser capaz de colaborar com o nosso projeto?

Título e descrição

Inclua o título e uma breve descrição sobre o projeto. Tente trazer clareza sobre o propósito do projeto.

Features

Feature é uma funcionalidade que o sistema se beneficia ou resolve algum problema do cliente. Enfatize! Cite as funcionalidades mais relevantes do seu projeto.

Tecnologias utilizadas

Ao longo da carreira, você irá criar muitos projetos. Hoje, você acabou de criar um projeto com C#. Será que daqui a alguns anos você lembrará quais foram as ferramentas utilizadas? Utilize o readme a seu favor e inclua as tecnologias empregadas no projeto, justificando a escolha e o objetivo da ferramentas.

Organização do projeto

Como o seu projeto está estruturado? Quais foram as pastas que contemplam os exercícios de programação e orientação a objetos? Indicar a forma como seu projeto está organizado, auxilia na visualização para que outras pessoas possam encontrar de maneira mais objetiva uma informação.

Pré-requisitos de instalação

Caso você esteja utilizando alguma ferramenta específica, tecnologia ou até mesmo o GitHub, indique que são pré-requisitos para a pessoa instalar antes de contribuir. Imagine que ruim seria a pessoa não conseguir evoluir a aplicação, porque não tem as ferramentas necessárias?

Execução da aplicação

Cada aplicação é executada de uma maneira. Enquanto no C# para executar o projeto você roda um comando dotnet run, em node é: npm start.

Erros comuns

Você passou por algum problema ao instalar uma ferramenta? Existe algum cenário conhecido que possa ser compartilhado?

Compartilhe experiências! Caso outra pessoa passe pelo mesmo erro, ela terá um caminho sobre como prosseguir para não ficar travada.

Contribuidores

Quais foram os contribuidores que auxiliaram no desenvolvimento do projeto? Caso a pessoa tenha algum problema ou queira fornecer um feedback, indique como ela pode se comunicar com você.

Dica!

Nem todos os itens dessa lista são obrigatórios. Quanto mais completo for o seu README, melhor será o entendimento para as pessoas que não fazem parte do projeto e que queiram contribuir.



Outra dica: use o idioma inglês se possível. É comum encontrar projetos com a descrição e o código fonte escrito em inglês, pois ele é muito usado e isso ajuda na expansão de seu projeto.

Confira a seguir um exemplo completo do código fonte de um arquivo readme.md.

```
1. zoom_in
2. # Projeto Exemplo
3. Repositório criado com o objetivo de compartilhar conteúdo sobre
orientação a
4. objetos.
5. ## :rocket: Descrição
6. "A programação orientada a objetos..."
7. ---
8. ## Tecnologias Utilizadas
9. Esse projeto foi criado utilizando as tecnologias:
10. ### Back-End
11. - [C#] (https://docs.microsoft.com/pt-br/dotnet/csharp/)
12. - [.NET] (https://dotnet.microsoft.com/download)
13. ### Editor
14. - [Visual Studio Code] (https://code.visualstudio.com/)
15. ---
16. ## Como rodar o projeto
17. Clone o projeto com o comando abaixo:
18. ````bash
19. # Clone o repositório
20. >git clone https://github.com/[usuario]/[nome-projeto].git
21.     # Entre no diretório
22.     >cd [nome-projeto]
23.         # Execute o projeto
24.         >dotnet run
25. `````
26. ---
27. ## Funcionalidades Futuras
28. - [x] A
29. - [ ] B
30. - [ ] C
31. ---
```

Confira a seguir o preview desse exemplo.

Projeto Exemplo

Repositório criado com o objetivo de compartilhar conteúdo sobre orientação a objetos.

:rocket: **Descrição**
"A programação orientada a objetos..."

 **Tecnologias Utilizadas**
Esse projeto foi criado utilizando as tecnologias:

- Back-End
 - C#
 - .NET

 **Editor**
• Visual Studio Code

 **Como rodar o projeto**
Clone o projeto com o comando abaixo:

```
# Clone o repositório
> git clone https://github.com/[usuario]/[nome-projeto].git

# Entre no diretório
> cd [nome-projeto]

# Execute o projeto
> dotnet run
```

 **Funcionalidades Futuras**
• [x] A
• [] B
• [] C

GITHUB

Depois de concluir o seu arquivo Readme.md, é importante disponibilizá-lo no repositório online. Para isso, uma das ferramentas mais utilizadas por empresas e desenvolvedores do mundo inteiro é o GitHub.

O GitHub é uma plataforma de hospedagem de código fonte e arquivos com controle de versão usando o Git. Com o GitHub e o Git podemos criar repositórios públicos (acesso irrestrito) e privados (somente pessoas autorizadas) para hospedar projetos e criar README's para descrever algumas funcionalidades e arquitetura técnica do projeto.



Fonte: Pixabay

PENSE NISSO...

Então, como colaborar em um projeto existente?

Através de uma documentação, você pode ter a compreensão sobre os itens do projeto, tecnologias, features disponíveis.

PDF

Para retomar o básico sobre GitHub, acesse seu material complementar.

Arquivo: 02_github.pdf

NESTE DESAFIO...

CODIFICAÇÃO BACK-END | DESAFIO 3



Neste desafio, você viu como documentar a construção do seu projeto técnico através do arquivo `readme.md`. Este arquivo, apesar de não ser essencial para o funcionamento do código, é de extrema importância para o projeto, pois explica de forma global e sucinta todo o projeto, e serve de base para inserir novos colaboradores ou para atualizações posteriores.

PARA CONCLUIR...

PARABÉNS, VOCÊ CONCLUIU UMA ETAPA DE CODIFICAÇÃO BACK-END!

Nessa etapa, abordamos conceitos básicos de server-side para poder utilizar a linguagem de programação C#. Depois de reconhecer todos os requisitos necessários

do C#, vimos como aplicar recursos da linguagem de programação para armazenar e manipular dados. Estudamos também o .NET, framework compatível com o C#, para otimizar o desenvolvimento.

Conhecemos paradigmas de programação e seus fundamentos, e nos aprofundamos na programação orientada a objetos, que é o paradigma principal do C#, para assim poder criar códigos reutilizáveis.

Depois reconhecemos a importância da documentação do projeto, e como criá-lo, para assim manter o projeto pronto para consultas e atualizações.

Continue estudando e se aprimorando. Até breve e sucesso!

REFERÊNCIAS

GREENE, Jennifer; STELLMAN, Andrew. **Head First C#.** 4a. edição. 800 pag. Newton: O'Reilly Media, 2007.

MDN WEB DOCS. **Introdução ao lado servidor.** Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Introduction. Acesso em: 11 de jun. de 2021.

MICROSOFT. **Documentação do C#.** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp>. Acesso em: 11 de jun. de 2021.

MICROSOFT. **List<T>** Classe. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/api/system.collections.generic.list-1?view=net-5.0>. Acesso em: 11 de jun. de 2021.

MICROSOFT. **O que é “código gerenciado”?** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/standard/managed-code>. Acesso em: 11 de jun. de 2021.

MICROSOFT. **Object-Oriented programming (C#).** Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop>. Acesso em: 11 de jun. de 2021.

MICROSOFT. **Programação funcional versus programação imperativa (LINQ to XML).** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/standard/linq/functional-vs-imperative-programming>. Acesso em: 11 de jun. de 2021. Disponível em: . Acesso em: 11 de jun. de 2021.

MICROSOFT. **Visual Studio Community.** Disponível em: <https://visualstudio.microsoft.com/pt-br/vs/community>. Acesso em: 11 de jun. de 2021.

MICROSOFT. **What is .NET Framework?** Disponível em: <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework>. Acesso em: 11 de jun. de 2021.

W3SCHOOLS. **C# Tutorial.** Disponível em: <https://www.w3schools.com/cs/index.php>. Acesso em: 11 de jun. de 2021.

Créditos

CONFEDERAÇÃO NACIONAL DA INDÚSTRIA -CNI

Robson Braga de Andrade
Presidente

DIRETORIA DE EDUCAÇÃO E TECNOLOGIA - DIRET

Rafael Esmeraldo Lucchesi Ramacciotti
Diretor de Educação e Tecnologia

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL – SENAI Conselho Nacional

Robson Braga de Andrade
Presidente

SENAI - Departamento Nacional

Rafael Esmeraldo Lucchesi Ramacciotti
Diretor-Geral

Gustavo Leal Sales Filho
Diretor de Operações

SENAI – DEPARTAMENTO NACIONAL UNIDADE DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA – UNIEP

Felipe Esteves Morgado
Gerente Executivo

Luiz Eduardo Leão
Gerente de Tecnologias Educacionais

*Anna Christina Theodora Aun de Azevedo
Nascimento*
Adriana Barufaldi
Bianca Starling Rosauro de Almeida
Laise Caldeira Pedroso
Coordenação Geral de Desenvolvimento dos
Recursos Didáticos Nacionais

SENAI - DEPARTAMENTO REGIONAL DE SÃO PAULO

Ricardo Figueiredo Terra
Diretoria Regional

Cassia Regina Souza da Cruz
Gerência de Educação

Izabel Rego de Andrade
Supervisão do Centro SENAI de Tecnologias
Educacionais

Claudia Baroni Savini Ferreira
Coordenação do Desenvolvimento do Curso

Helena Strada Franco de Souza
Elaboração de Conteúdo

Adilson Moreira Damasceno
Orientação de Práticas de Educação a Distância

Paula Cristina Bataglia Buratini
Coordenação da Produção do Curso

Cristina Yurie Takahashi
Katya Martinez Almeida
Design Educacional

Luana Dorizo de Melo
Diagramação

Cleriston Ribeiro de Azevedo
Fabiano José Moura
Juliana Rumi Fujishima
Ilustrações

Camila Ciarini Dias
Produção e Edição de Vídeos

Rafael Santiago Apolinário
Programação

Aldo Toma Junior
Web Design