Construtores POO com C#

SW-I – Prof. Anderson Vanin

O que é um construtor?

Construtores são basicamente **funções de inicialização de uma classe**, as quais são invocadas no momento em que objetos desta classe são criadas. Eles permitem inicializar campos internos da classe e alocar recursos que um objeto da classe possa demandar, tais como memória, arquivos, semáforos, soquetes, etc.

Agora imagine que você tenha uma classe Cliente e seu atributo nome. Como vimos nas aulas anteriores, nós precisamos lembrar de colocar o nome após criarmos um novo cliente em nosso sistema. Isso pode ser visto no código a seguir:

```
Cliente cliente = new Cliente();

cliente.Nome = “Anderson";
```

E se esquecermos de chamar a segunda linha desse código, teremos um cliente sem nome.

Mas, será que faz sentido existir um cliente sem nome?

Construtor

Para evitar isso, ao construir nosso objeto temos que obrigar o desenvolvedor a falar qual o nome do Cliente. Isto é, queremos ser capazes de alterar o comportamento da construção do objeto.

Queremos definir um novo comportamento que dirá como será construído o objeto. Algo como:

```
Cliente cliente = new Cliente(“Anderson Vanin");
```

Note que esse comportamento que desejamos lembra um comportamento normal, passando argumentos, mas com a característica especial de ser quem constrói um objeto. Esse comportamento recebe o nome de construtor. E como defini-lo? Similarmente a um comportamento qualquer:

Construtor

```
class Cliente

{

    // Outros atributos da classe Cliente

    public string Nome { get; set; }

    public Cliente (string nome)

    {

        this.Nome = nome;

    }

}
```

Construtor

Vimos que quando criamos um construtor na classe, o C# usa o construtor criado para inicializar o objeto, porém o que acontece quando não temos nenhum construtor na classe? Quando uma classe não tem nenhum construtor, o C# coloca um construtor padrão dentro da classe. Esse construtor não recebe argumentos e não executa nenhuma ação, ou seja, um construtor que não recebe nenhum argumento e tem o corpo vazio.

Múltiplos construtores dentro da classe

Na seção anterior definimos um construtor dentro da classe cliente que inicializa a propriedade nome, mas e se quiséssemos inicializar também a idade do Cliente durante a construção do objeto? Nesse caso, precisaríamos de um construtor adicional na classe Cliente:

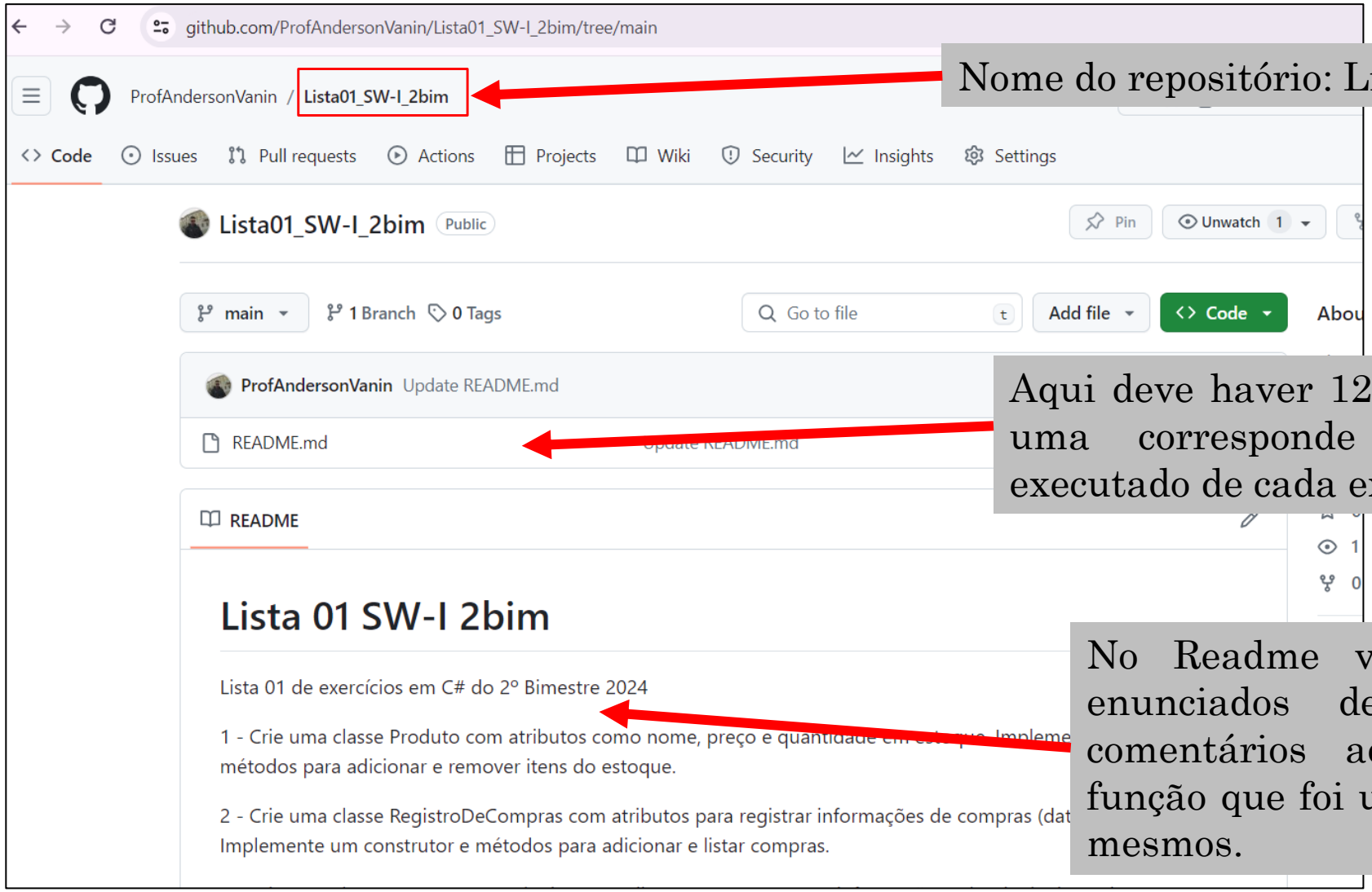
```
class Cliente {
    public string Nome { get; set; }
    public int Idade { get; set; }
    // construtor que só recebe o nome
    public Cliente (string nome)
    {
        this.Nome = nome;
    }
    // construtor que recebe o nome e a idade
    public Cliente (string nome, int idade)
    {
        this.Nome = nome;
        this.Idade = idade;
    }
}
```

EXERCÍCIOS

Para a resolução destes exercícios, **você pode usar o chatgpt** para a sugestão de respostas. Modifique a seu critério e **ENTENDA CADA SITUAÇÃO**. **Sua avaliação nesta atividade será com base no funcionamento de cada ítem aplicado à resolução desta tarefa!**

Implemente todos os exercícios em um **repositório do github chamado Lista01_SW-I_2bim**. Crie um projeto novo para cada exercício, renomei-os como **exe01, exe02**, etc.

EXERCÍCIOS

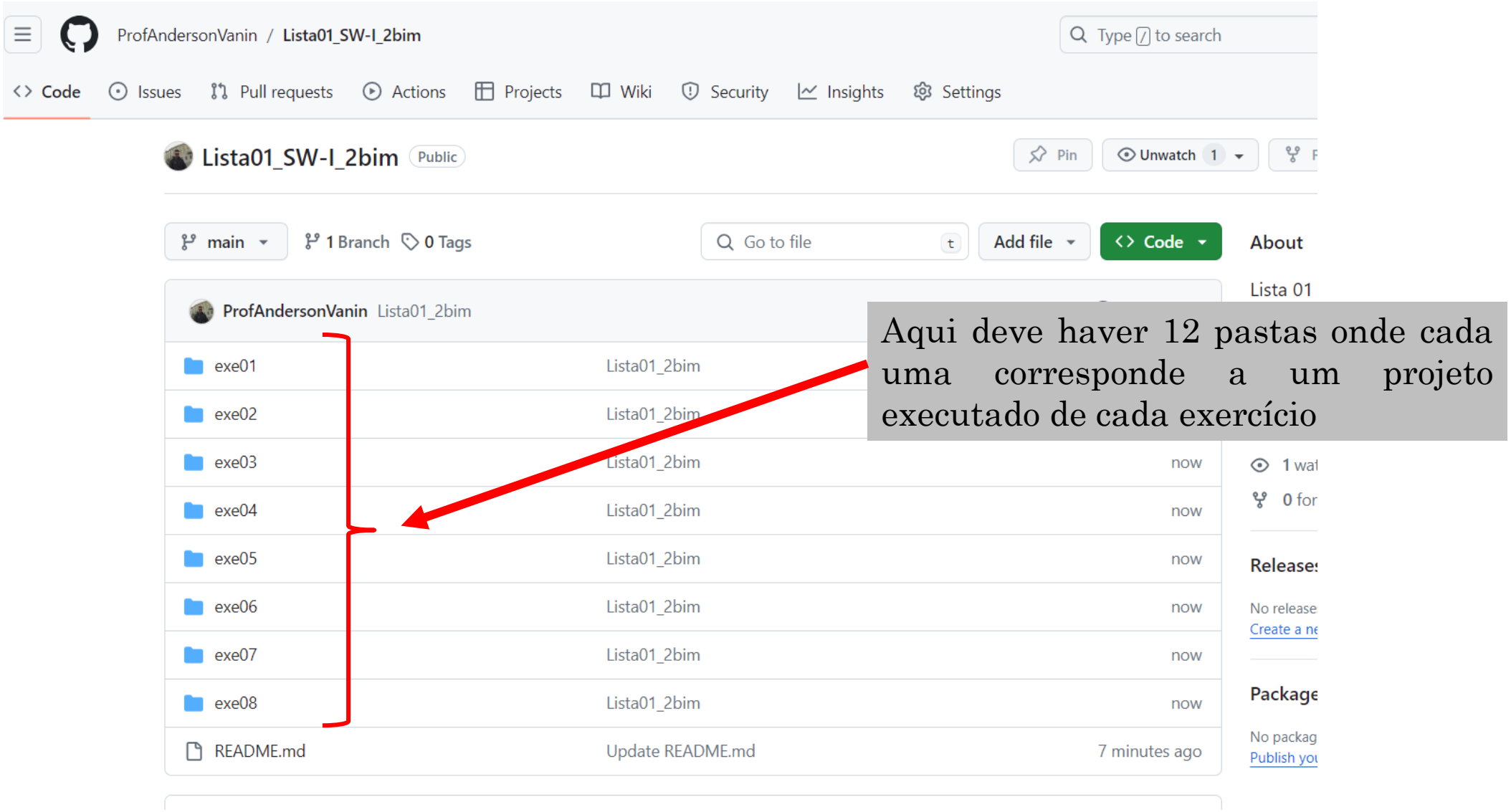


Nome do repositório: Lista01_SW-I_2bim

Aqui deve haver 12 pastas onde cada uma corresponde a um projeto executado de cada exercício

No Readme você pode colocar os enunciados de cada exercício e comentários adicionais sobre cada função que foi usada na resolução dos mesmos.

EXERCÍCIOS


ProfAndersonVanin / Lista01_SW-I_2bim
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Lista01_SW-I_2bim
Public
Pin
Unwatch 1
main
1 Branch
0 Tags
Go to file
t
Add file
Code
About
Lista 01
ProfAndersonVanin Lista01_2bim
exe01 Lista01_2bim
exe02 Lista01_2bim
exe03 Lista01_2bim now
exe04 Lista01_2bim now
exe05 Lista01_2bim now
exe06 Lista01_2bim now
exe07 Lista01_2bim now
exe08 Lista01_2bim now
README.md Update README.md 7 minutes ago
Aqui deve haver 12 pastas onde cada uma corresponde a um projeto executado de cada exercício
1 wat
0 for
Releases
No release
Create a ne
Package
No packag
Publish you

Exercícios

1) Crie uma classe Produto com atributos como nome, preço e quantidade em estoque. Implemente um construtor e métodos para adicionar e remover itens do estoque.

2) Crie uma classe RegistroDeCompras com atributos para registrar informações de compras (data, produto, valor). Implemente um construtor e métodos para adicionar e listar compras.

3) Crie uma classe CorretoraDeImóveis com atributos para armazenar informações sobre imóveis (endereço, preço, tipo). Implemente um construtor e métodos para listar imóveis disponíveis e calcular o valor médio. Crie um menu que contenha as opções de inserir imoveis, alterar preço do imovel e apresentar todos os imoveis. Insira os imoveis numa lista para facilitar o trabalho.

4) Crie uma classe AgendaTelefônica com atributos para armazenar contatos (nome, telefone, e-mail). Implemente um construtor e métodos para adicionar numa lista, remover e buscar contatos.

Exercícios

5) Crie uma classe GerenciadorDeTarefas com atributos para armazenar uma lista de tarefas (descrição, data de vencimento). Implemente um construtor e métodos para adicionar, remover e listar tarefas. Adicione um método para verificar se a tarefa deverá ser executada no dia de hoje.

6) Utilize o exercício 1 para criar uma lista de produtos e faça métodos para apresentar na tela e para consultar produtos pelo código.

7) Crie uma classe chamada Música que contenha os atributos nome, autor e gravadora. Após, crie uma classe Playlist que possua como atributo uma Lista de músicas e uma string para armazenar o dono da playlist. Implemente um método para adicionar músicas na lista, para "tocar a música" (só mostrar uma mensagem na tela com o titulo da música). Tente fazer uma reprodução aleatória, ou seja, em vez de percorrer do inicio ao fim da lista, faça um random para acessar valores aleatórios válidos.

Exercícios

8) Crie classes chamadas Fabricante e Produto. Fabricante que tenha as propriedades Nome, Endereço e Cidade. Produto que tenha as propriedades Nome, Fabricante (objeto da classe Fabricante) e Preco. Utilize o encapsulamento para garantir que o nome não seja vazio e que o preço seja positivo.

9) Crie uma classe chamada Livro que tenha as propriedades Titulo e Autor. Utilize o encapsulamento para garantir que o título e o autor não sejam vazios.

10) Crie uma classe chamada Animal que tenha as propriedades Nome, Especie e Idade. Crie também um método chamado EmitirSom, que imprime na tela o som do animal.

Exercícios

11) Crie uma classe "Carro" com os atributos "modelo", "ano" e "velocidade". Em seguida, crie um método para acelerar o carro (aumentando a velocidade em 10) e outro para frear o carro (diminuindo a velocidade em 10, mas nunca deixando a velocidade negativa).

12) Crie uma classe Agenda que pode armazenar inúmeras pessoas e que seja capaz de realizar as seguintes operações:

- void armazenaPessoa(String nome, int idade, float altura);
- void removePessoa(String nome);
- Pessoa buscaPessoa(String nome); retorna a pessoa com o nome informado
- void imprimeAgenda(); imprime os dados de todas as pessoas da agenda

Faça um menu para navegar entre as opções infinitamente.

Resposta - 01

```
using System;
public class Produto
{
    // Atributos
    private string nome;
    private double preco;
    private int quantidadeEmEstoque;

    // Construtor
    public Produto(string nome, double preco, int quantidadeInicial)
    {
        this.nome = nome;
        this.preco = preco;
        this.quantidadeEmEstoque = quantidadeInicial;
    }
    // Método para adicionar itens ao estoque
    public void AdicionarEstoque(int quantidade)
    {
        if (quantidade > 0)
        {
            quantidadeEmEstoque += quantidade;
            Console.WriteLine($"{quantidade} unidades do produto '{nome}' foram adicionadas ao estoque.");
        }
        else
        {
            Console.WriteLine("A quantidade a ser adicionada deve ser maior que zero.");
        }
    }
    // Método para remover itens do estoque
    public void RemoverEstoque(int quantidade)
    {
        if (quantidade > 0 && quantidade <= quantidadeEmEstoque)
        {
            quantidadeEmEstoque -= quantidade;
            Console.WriteLine($"{quantidade} unidades do produto '{nome}' foram removidas do estoque.");
        }
        else if (quantidade > quantidadeEmEstoque)
        {
            Console.WriteLine($"Não há {quantidade} unidades do produto '{nome}' disponíveis no estoque.");
        }
        else
        {
            Console.WriteLine("A quantidade a ser removida deve ser maior que zero.");
        }
    }
```

Resposta - 01

```
// Método para exibir informações do produto
    public void MostrarInformacoes()
    {
        Console.WriteLine($"Nome: {nome}");
        Console.WriteLine($"Preço: R${preco:F2}");
        Console.WriteLine($"Quantidade em estoque: {quantidadeEmEstoque}");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        // Exemplo de uso da classe Produto
        Produto produto1 = new Produto("Camiseta", 29.99, 50);
        produto1.MostrarInformacoes();

        produto1.AdicionarEstoque(10);
        produto1.RemoverEstoque(5);
        produto1.RemoverEstoque(60); // Tentativa de remover mais itens do que disponível

        Console.WriteLine("\nAtualizando informações do produto após operações:");
        produto1.MostrarInformacoes();
    }
}
```