Nome: Náthan Luciano RA:043412

1) Classe Anêmica na Programação: Refere-se a uma classe que possui principalmente atributos e métodos simples de acesso aos dados, sem lógica complexa ou funcionalidades avançadas.

Exemplo de Classe Anêmica: Uma classe 'Pessoa' com atributos como nome, idade e gênero, juntamente com métodos de acesso simples a esses dados, mas sem funcionalidades complexas.

Classe Rica na Programação: Refere-se a uma classe que contém atributos, além de métodos robustos que encapsulam lógica complexa e funcionalidades importantes.

Exemplo de Classe Rica: Uma classe 'ContaBancaria' que não apenas armazena saldo e titular, mas também possui métodos complexos para operações bancárias, como depósito, saque e transferência, com lógica detalhada.

2) O Princípio da Responsabilidade Única (SRP) no SOLID estabelece que uma classe deve ter apenas uma razão para mudar, ou seja, uma única responsabilidade bem definida. Isso é importante para manter o código claro, modular e de fácil manutenção.

using System;

// Classe para lidar com operações de armazenamento de dados do produto

public class Product

{

public string Name { get; set; }

public decimal Price { get; set; }

public int Quantity { get; set; }

}

// Classe para gerar relatórios com base nos dados do produto

public class ProductReport

{

public static string GenerateReport(Product product)

{

return $"Product Report:\nName: {product.Name}\nPrice: {product.Price}\nQuantity: {product.Quantity}\n";

}

}

class Program

{

static void Main(string[] args)

{

// Criando um objeto da classe Product

Product product1 = new Product

{

Name = "Laptop",

Price = 1200,

Quantity = 10

};

// Utilizando a classe Product para armazenar os dados do produto

// e a classe ProductReport para gerar um relatório com base nesses dados

Console.WriteLine("Informações do Produto:");

Console.WriteLine($"Name: {product1.Name}, Price: {product1.Price}, Quantity: {product1.Quantity}\n");

string report = ProductReport.GenerateReport(product1);

Console.WriteLine("Relatório do Produto:");

Console.WriteLine(report);

}

}

3) Integridade: Garante a consistência e precisão dos dados, evitando inconsistências que podem afetar a funcionalidade do sistema.

Segurança: Previne vulnerabilidades e ataques, mantendo os dados seguros contra injeções de código e outras ameaças.

Experiência do Usuário: Contribui para uma melhor experiência, evitando erros e fornecendo orientação clara durante a entrada de dados.