IPCA INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

Trabalho Prático

Ana Paula Canuto da Silva, 24178

Programação Orientada a Objetos

Prof. Luís Ferreira & Ernesto Casanova

Licenciatura em Engenharia em Sistemas Informáticos

(regime pós-laboral)

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

Data do documento: 16 de novembro de 2025

Índice

Introdução

O presente relatório descreve a estrutura e o funcionamento inicial do projeto **Gestão de Alojamentos Turísticos**, desenvolvido em **C#** utilizando **Programação Orientada a Objetos (POO)**. O objetivo do sistema é gerenciar informações de clientes e alojamentos, permitindo o cálculo de taxas e a classificação de hotéis, servindo como base para futura integração com interface gráfica e funcionalidades avançadas de reservas e relatórios.

Relatório Técnico – Projeto POO: Gestão de Alojamentos Turísticos

1- Visão Geral do Projeto

O projeto tem como objetivo criar uma aplicação orientada a objetos para gerenciar alojamentos turísticos e clientes, com possibilidade de expansão para incluir reservas, faturação, relatórios e interface gráfica (Forms).

Atualmente, o projeto contém três classes principais:

- **Alojamento:** classe base que representa um alojamento genérico.
- **Cliente:** classe que representa o cliente do sistema.
- **Hotel:** classe derivada de Alojamento, que adiciona o conceito de classificação por estrelas.

2- Estrutura das Classes

Diagrama UML – Gestão de Alojamentos Turísticos, exemplificando a estrutura das 3 classes criadas.

Explicação:

- Alojamento é a classe base, com dados do alojamento e método de cálculo de taxa.
- Hotel herda de Alojamento e adiciona Número de Estrelas + métodos específicos.
- Cliente é independente, representando os clientes do sistema.

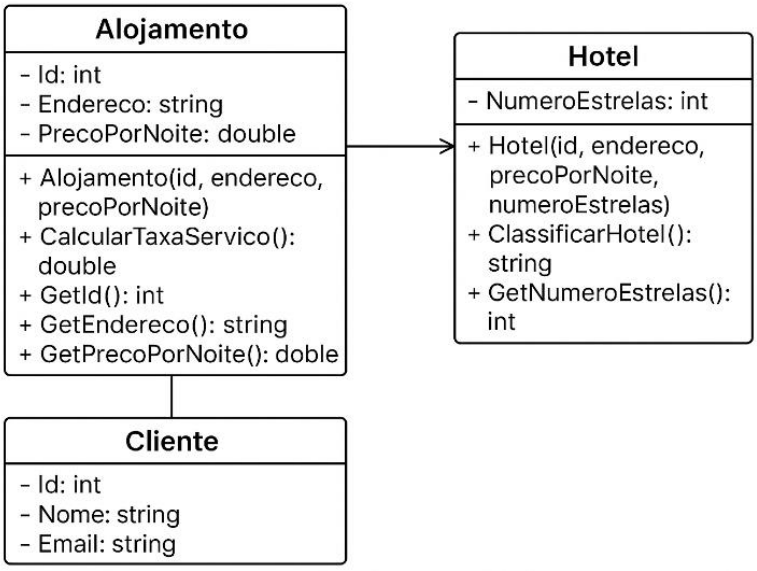

Figura 1 Diagrama_UML das_Classes

2.1 Classe Alojamento

Propriedades privadas: Id, Endereco, PrecoPorNoite.

Construtor protegido: permite que classes derivadas (como Hotel) possam inicializar a instância.

Métodos públicos:

- CalcularTaxaServico() – retorna 10% do preço por noite.
- GetId(), GetEndereco(), GetPrecoPorNoite() – getters públicos para leitura segura.

Pontos fortes:

- Mantém **encapsulamento** adequado.
- Permite **herança segura**, com o construtor protegido.
- Métodos públicos fornecem funcionalidades básicas sem expor dados diretamente.

Melhoria futura que pretendo fazer:

- Avaliar uso de **propriedades auto-implementadas** com get público e set privado, que é mais idiomático em C#.
- Considerar métodos adicionais, como AtualizarPreco(double novoPreco) para manter consistência de dados.

2.2 Classe Cliente

- **Propriedades privadas:** Id, Nome, Email.
- **Construtor público:** inicializa os dados do cliente.
- **Método público:** ValidarEmail() para checar se o email contém @.

Pontos fortes:

- Boas práticas de encapsulamento.
- Método de validação simples implementado.

Pretendo melhorar:

- Adicionar **getters públicos** para leitura segura do cliente (GetNome(), GetEmail()), caso o Forms ou outras partes do sistema precisem acessar essas informações.
- Implementar **validações adicionais**, como formato de email correto ou campos obrigatórios.

- Possibilidade de implementar uma **classe ou interface de contato** que englobe email, telefone, etc.

2.3 Classe Hotel (herda de Alojamento)

- **Propriedade privada:** NumeroEstrelas.
- **Construtor público:** chama base() para inicializar Alojamento.
- **Método público:** ClassificarHotel() retorna uma string (Luxo, Conforto, Standard) com base no número de estrelas.
- **Getter público:** GetNumeroEstrelas().

Pontos fortes:

- Demonstra **uso correto de herança**.
- Mantém **encapsulamento** adequado das propriedades.
- Métodos públicos fornecem funcionalidade derivada de maneira clara e intuitiva.

Possíveis melhorias futuras:

- Adicionar funcionalidades específicas de hotéis, como:
  - ✓ Número de quartos disponíveis.
  - ✓ Serviços oferecidos (piscina, restaurante, Wi-Fi).
  - ✓ Gestão de reservas e ocupação.
- Avaliar criação de outras subclasses de Alojamento, como Apartamento, Hostel ou AlojamentoRural, para expandir o sistema.

3- Boas Praticas Praticadas

- **Encapsulamento:** propriedades privadas + getters públicos.
- **Herança:** Hotel estende Alojamento corretamente.
- **Construtor protegido:** garante inicialização controlada em heranças.
- **Métodos públicos para funcionalidades essenciais:** mantém coesão da classe.

4 – Planejamento dos Próximos Passos

4.1 Interface Gráfica (Forms)

Criar formulários separados para:

- Cadastro de clientes.
- Cadastro e gestão de alojamentos.
- Visualização de hotéis e suas classificações.

Integrar métodos existentes (GetEndereco(), ClassificarHotel()) para exibir dados no UI sem quebrar encapsulamento.

4.2 Gestão de Reservas

Criar classe Reserva que relacione Cliente e Alojamento:

- Datas de check-in/check-out.
- Número de pessoas.
- Valor total da estadia (usando PrecoPorNoite e CalcularTaxaServico()).

4.3 Validações Avançadas

- Email, telefone e campos obrigatórios para Cliente.
- Preço por noite e número de estrelas para Alojamento e Hotel.

4.4 Expansão do Modelo de Herança

Outras subclasses de Alojamento (ex: Apartamento, Hostel) usando polimorfismo para métodos como CalcularTaxaServico().

4.5 Persistência de Dados

Considerar salvar informações em arquivos (JSON, XML) ou banco de dados (SQL Lite ou SQL Server).

4.6 Implementação de testes

Implementação de testes para avaliar a execução da solução desenvolvida, unitários e não unitários.

Conclusão e Trabalhos Futuros

O projeto apresenta uma base sólida de **Programação Orientada a Objetos**, com classes bem estruturadas, encapsulamento adequado, herança corretamente aplicada e métodos públicos apropriados. As funcionalidades atuais oferecem uma fundação consistente para futuras expansões, como a integração com uma interface Forms, gestão de reservas, clientes e relatórios, além de persistência de dados. Com pequenas melhorias, como a adição de getters, validações e subclasses extras, será possível desenvolver uma aplicação robusta, escalável e organizada, mantendo boas práticas de desenvolvimento e garantindo facilidade na evolução do sistema.

Referências Bibliográficas

- Repositório do professor Ernesto Casanova em: https://github.com/IPCA-Content/POO-LESI-PL-202526/tree/main/Lessons
- Conteúdos sobre C# em Alura Cursos: https://www.alura.com.br/
- ChatGPT (Duvida sobre para corrigir métodos públicos) https://chatgpt.com/

Apêndice

Figura	**Descrição**
Figura 1	Diagrama UML das Classes