Instituto Politécnico do Cavado e Ave

Ano Letivo: 2025/2026

Trabalho Prático

Programação Orientada Objetos

Gestão de Obras de Construção Civil

Professor:

Luís Ferreira

Nº/Aluno:

31501 - Tomás Afonso Cerqueira Gomes

Data:

Dezembro de 2025

Índice

Índice Imagens

Introdução

Enquadramento

O projeto de Gestão de Obra de Construção Civil visa a criação de um sistema para gerir os aspetos logísticos e operacionais de uma obra de construção civil. As obras de construção envolvem uma série de elementos que precisam ser geridos de forma eficiente, como materiais, serviços, mão de obra, fornecedores e orçamentos. A digitalização e automação desses processos podem reduzir custos e melhorar a eficiência na execução das obras.

Motivação

Este trabalho tem como objetivo desenvolver uma aplicação em linguagem C# que utiliza o paradigma de Programação Orientada a Objetos (POO) para gerir os recursos e custos associados a obras de construção civil. O projeto é dividido em duas fases, com diferentes níveis de abstração e complexidade.

Fase 1 – Modelação e Estrutura de Classes

Nesta fase, o foco principal recaiu sobre a transposição do problema real para código, através da criação de classes que representam as entidades do domínio (Obras, Materiais, Funcionários, etc.). A motivação foi estabelecer uma hierarquia de classes robusta, aplicando pilares da POO como a Herança e o Polimorfismo, permitindo tratar diferentes tipos de custos de forma unificada.

Fase 2 – Persistência, Regras de Negócio e Robustez

A segunda fase do projeto introduz a arquitetura em camadas e a persistência de dados. A motivação evoluiu para a criação de um sistema funcional e seguro: os dados passam a ser guardados em ficheiros binários, impedindo a perda de informação, e são implementadas regras de validação rigorosas (camada de Regras) e tratamento de erros (Exceções personalizadas) para garantir a integridade do sistema.

Objetivos

O objetivo principal deste trabalho é desenvolver um sistema modular capaz de gerir o ciclo de vida de uma obra, calculando custos totais e garantindo a persistência da informação entre execuções.

Objetivos da Fase 1:

- Modelar o problema utilizando diagramas UML e transpô-lo para linguagem C#.
- Implementar as classes base (Business Objects) com recurso a herança (ex: AbsObra) e interfaces (ICusto).
- Aplicar o encapsulamento para proteger os dados das entidades.
- Estruturar o projeto de forma a permitir extensibilidade futura.

Objetivos da Fase 2:

- Implementar uma arquitetura N-Tier (Camadas), separando Dados, Regras, Objetos (BO) e Interface.
- Desenvolver mecanismos de persistência de dados utilizando serialização em ficheiros binários.
- Criar e implementar Exceções Personalizadas (FicheiroException) para um tratamento de erros eficaz.
- Implementar validações lógicas (Regras de Negócio) para impedir a inserção de dados incoerentes.
- Validar o funcionamento do código através de Testes Unitários (MSTest).

Metodologia de trabalho

Este trabalho foi desenvolvido por etapas, respeitando uma evolução natural desde a modelação teórica até à implementação técnica robusta.

Fase 1: Começou-se pela análise dos requisitos e desenhou-se o Diagrama de Classes para definir as relações entre as entidades. Desenvolveram-se as classes de domínio (BO), focando na correta aplicação de herança e polimorfismo. A lógica inicial foi testada através da instanciação simples de objetos e verificação dos cálculos de custos em memória.

Fase 2: Após validar a estrutura de classes, avançou-se para a implementação das camadas funcionais. Criou-se a camada de Dados para gerir a leitura e escrita em ficheiros e a camada de Regras para validar a entrada de dados. Posteriormente, integrou-se o tratamento de exceções para blindar a aplicação contra erros de execução. Por fim, utilizou-se o projeto de Testes Unitários para garantir que as regras de negócio estavam a funcionar corretamente antes da integração final na aplicação de consola.

Trabalho desenvolvido

Analise e especificação

Requisitos

Fase 1: O objetivo primário da primeira fase foi a transposição do problema real para um modelo orientado a objetos. O sistema foi desenhado para permitir a criação e manipulação em memória dos principais elementos de uma obra. As funcionalidades base implementadas foram:

- **Gestão de Obras:** Criação de obras com nome, localização e intervalo temporal (início e término).
- **Gestão de Recursos:** Adição de **Materiais** (nome, descrição, custo) e **Serviços** (descrição, custo).
- **Gestão de Recursos Humanos:** Registo de **Funcionários** com nome, função e cálculo de salário base.
- **Gestão de Fornecedores:** Registo simples de **Fornecedores** e respetivos contactos.
- **Cálculo de Orçamentos:** Capacidade de agregar os custos de todos os elementos anteriores para obter o custo total da obra.

Fase 2: Na segunda fase, os requisitos evoluíram para garantir a durabilidade dos dados e a estabilidade da aplicação. Novas funcionalidades introduzidas:

- **Persistência de Dados:** Capacidade de gravar e ler o estado da aplicação (obras, catálogos de materiais, etc.) em ficheiros binários, garantindo que a informação não se perde ao fechar o programa.
- **Validação de Regras de Negócio:** Implementação de verificações lógicas antes da inserção de dados (ex: impedir custos negativos ou nomes vazios).
- **Tratamento de Erros:** Sistema robusto de exceções para lidar com falhas de ficheiros ou dados inválidos sem crashar a aplicação.

.

Arquitetura Lógica/Funcional

A arquitetura do sistema baseia-se nos princípios da abstração e encapsulamento, típicos da Programação Orientada a Objetos, tendo evoluído de uma estrutura monolítica para uma arquitetura em camadas (N-Tier).

Fase 1: Estrutura de Classes

Nesta fase, definiu-se o modelo de dados (agora situado na biblioteca de classes **BO - Business Objects**). Cada classe gere as suas informações de forma

independente, promovendo a modularidade. As principais classes desenvolvidas foram:

- **AbsObra:** Classe abstrata que define a estrutura base (nome, localização, datas), servindo de modelo para extensões futuras.
- **AbsOrcamento:** Classe abstrata responsável pela definição dos elementos essenciais de custos.
- **Material e Servico:** Representam os recursos físicos e tarefas contratadas, respetivamente, contendo descrições e custos unitários.
- **Funcionario:** Modela os trabalhadores, encapsulando a lógica de cálculo de pagamento.
- **Fornecedor:** Representa as entidades externas fornecedoras de recursos.

Fase 2: Arquitetura em Camadas (N-Tier)

Para cumprir os requisitos de organização e persistência, o projeto foi reestruturado em camadas lógicas distintas:

- **BO (Business Objects):** Contém apenas as entidades descritas na Fase 1 (Classes simples).
- **Dados (Data Access Layer):** Classes responsáveis exclusivamente pela gravação e leitura de ficheiros (ex: Materiais.cs, Obras.cs). Utiliza serialização binária.
- **Regras (Business Logic Layer):** Camada intermédia que valida os dados recebidos da aplicação antes de chamar a camada de Dados (ex: RegrasObras.cs verifica se a data de fim é superior à de início).
- **Exceções:** Projeto transversal contendo erros personalizados (ex: FicheiroException) para normalizar o tratamento de falhas.

Interação

Fase 1: Interação entre Objetos A interação inicial focava-se na comunicação direta entre as entidades para realizar cálculos em memória:

- A classe **Obra** atua como elemento central, agregando listas de Materiais, Serviços e Funcionários.
- O cálculo de custos é feito através de polimorfismo e iteração sobre estas listas, onde a Obra solicita o custo a cada elemento (Material, Serviço, Funcionário) e soma o total.
- O Program.cs instenciava diretamente os objetos e adicionava-os às listas da Obra.

Fase 2: Fluxo de Execução por Camadas Com a introdução da arquitetura em camadas, o fluxo de interação tornou-se mais estruturado e seguro:

1. **Interface (App):** O Program.cs recolhe os dados do utilizador e envia-os para a camada de **Regras**.
2. **Validação (Regras):** A camada de Regras verifica se os dados são válidos (ex: preço > 0). Se não forem, lança uma Exceção e devolve o erro à App.
3. **Persistência (Dados):** Se as regras forem cumpridas, a camada de Regras invoca a camada de **Dados** para gravar a informação em disco.
4. **Resposta:** A confirmação de sucesso percorre o caminho inverso até ao utilizador.

Implementação

A implementação do sistema foi realizada em linguagem **C#**, utilizando o ambiente de desenvolvimento **Microsoft Visual Studio**. Ao contrário de uma abordagem monolítica, a solução foi estruturada em múltiplos projetos (.NET Class Libraries), cada um representando uma camada lógica da aplicação.

Esta organização em "Solution" permitiu isolar as dependências e garantir que a camada de apresentação (Consola) não acede diretamente aos dados sem passar pelas regras de negócio. A solução final é composta pelos seguintes projetos:

- **BO (Business Objects):** Biblioteca de classes contendo as entidades base.
- **Dados:** Biblioteca responsável pela persistência em ficheiros.
- **Regras:** Biblioteca que contém a lógica de validação.
- **Exceções:** Biblioteca transversal para gestão de erros personalizados.
- **Testes:** Projeto de Testes Unitários (MSTest).
- **GestãoObraApp:** Aplicação de Consola (Frontend).

Fase 1: Modelação e Entidades (Camada BO)

Na primeira fase, o esforço de implementação focou-se na tradução do diagrama de classes para código C#. Todas as entidades foram agrupadas no projeto **BO**.

- **Classes Abstratas e Herança:** Implementou-se a classe AbsObra para definir o contrato base de qualquer obra (propriedades como Nome, Localizacao, DataInicio). A classe concreta Obra herda desta estrutura e implementa os métodos abstratos, garantindo a extensibilidade do sistema.

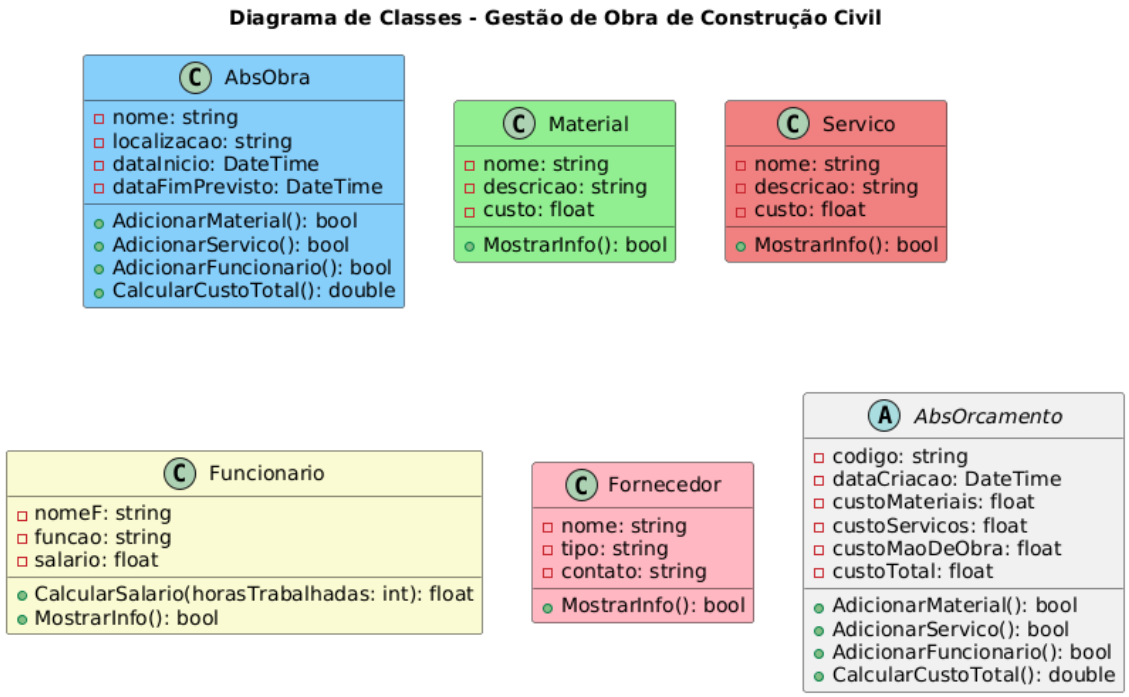

Diagrama UML – Classes

- **Encapsulamento:** Todos os atributos foram protegidos, sendo o acesso realizado exclusivamente através de Propriedades (Properties) com modificadores get e set, protegendo o estado interno dos objetos.
- **Interfaces e Polimorfismo:** Foi preparada a base para o cálculo de custos, onde diferentes entidades (Material, Serviço, Funcionário) podem responder de forma polimórfica ao pedido de cálculo do seu valor no orçamento total.

Fase 2: Camadas e Lógica de Negócio

A segunda fase consistiu na implementação da arquitetura "N-Tier" (Multicamada) para garantir a persistência e a robustez.

Para permitir que os dados sobrevivam ao fecho da aplicação, implementou-se a serialização.

- Foram criadas classes estáticas de gestão (Materiais.cs, Obras.cs, Funcionarios.cs) que mantêm listas em memória durante a execução.
- O método GravarFicheiro utiliza a classe BinaryFormatter para converter os objetos complexos em fluxos de bytes, guardando-os em ficheiros físicos (ex: obras.bin).
- Esta abordagem permite guardar o estado exato dos objetos, incluindo todas as suas relações e listas internas.

Implementou-se uma camada intermédia de "Regras" (ex: RegrasObras.cs, RegrasMateriais.cs) que atua como barreira de segurança.

- Nenhum dado chega à camada de Dados sem antes ser validado aqui.
- Foram implementadas validações lógicas, tais como: impedir preços negativos em materiais, exigir nomes não-nulos em funcionários e garantir que a data de fim de uma obra não é anterior à data de início.

Para melhorar o diagnóstico de erros, criou-se uma biblioteca dedicada com a classe FicheiroException.

- Em vez de apresentar erros genéricos do sistema operativo, o sistema captura o erro original e "embrulha-o" numa FicheiroException com uma mensagem amigável para o utilizador.
- Isto permite distinguir claramente entre erros de lógica de negócio (validações) e erros técnicos (falha no disco).

Para garantir a qualidade do código, foi adicionado um projeto de MSTest. Foram implementados testes unitários para verificar se as regras de negócio estão a funcionar como esperado (por exemplo, testar se o sistema lança efetivamente uma exceção ao tentar inserir um funcionário com salário negativo).

Analise e discussão dos resultados

O desenvolvimento deste projeto foi realizado de forma incremental e acompanhado por testes constantes, o que permitiu detetar e resolver problemas de arquitetura e lógica à medida que estes surgiam. A separação do projeto em duas fases distintas foi essencial para consolidar primeiro os conceitos de modelação de objetos e, posteriormente, a complexidade da persistência e validação de dados.

Fase 1: Na primeira fase, o foco principal esteve na correta aplicação dos pilares da Programação Orientada a Objetos.

- **Desafios:** A maior dificuldade prendeu-se com a definição correta da hierarquia de classes, especificamente na distinção entre o que deveria ser abstrato (AbsObra) e o que deveria ser concreto (Obra).
- **Resultados:** A implementação de herança e polimorfismo revelou-se eficaz. O método CalcularCustoTotal() demonstrou a flexibilidade do sistema: a classe Obra consegue iterar sobre listas de objetos diferentes (Materiais, Serviços, Funcionários) e somar os seus custos sem precisar de conhecer os detalhes de implementação de cada um.

Fase 2: A segunda fase trouxe um nível de complexidade superior, exigindo a reestruturação da solução para uma arquitetura em camadas (N-Tier).

- **Persistência de Dados:** A implementação da serialização binária apresentou desafios iniciais, nomeadamente a necessidade de garantir que todas as classes envolvidas (incluindo as contidas em listas) estivessem marcadas como [Serializable]. Os testes confirmaram que o uso de BinaryFormatter permite guardar e recuperar o estado complexo da obra (com todas as suas listas internas) de forma íntegra.
- **Validação e Regras:** A criação da camada intermédia (Regras) provou ser fundamental para a robustez do sistema. Os testes realizados demonstraram que é impossível inserir dados inválidos (como custos negativos ou nomes vazios) através da aplicação, pois a camada de regras interceta o pedido e lança uma exceção antes de chegar aos dados.
- **Tratamento de Erros:** A introdução de exceções personalizadas (FicheiroException) permitiu melhorar a experiência de utilização. Em vez de o programa "crashar" com erros técnicos do sistema operativo, o utilizador recebe mensagens claras sobre o que correu mal.

No geral, o sistema desenvolvido demonstrou-se robusto, modular e funcional. A adoção de uma arquitetura multicamada, embora mais trabalhosa inicialmente, resultou num código muito mais organizado: a camada de apresentação não

conhece a camada de dados, e a lógica de negócio está centralizada. Isto significa que futuras alterações (como mudar a gravação de ficheiros binários para uma base de dados SQL) poderiam ser feitas alterando apenas a camada de Dados, sem afetar o resto do programa. Os testes unitários realizados confirmaram a fiabilidade das validações implementadas, garantindo que o software cumpre os requisitos propostos com segurança e estabilidade.

Conclusão

O projeto de Gestão de Obras de Construção Civil foi concluído com sucesso, e as soluções desenvolvidas respondem em conformidade a todos os requisitos propostos no enunciado. O objetivo principal de criar um sistema capaz de gerir recursos, calcular orçamentos e persistir dados foi plenamente atingido.

Durante a primeira fase, o principal desafio foi a modelação correta do problema utilizando os princípios da Programação Orientada a Objetos. A definição de uma hierarquia de classes coerente, com o uso de classes abstratas (AbsObra), foi fundamental para garantir que o sistema fosse flexível e capaz de tratar diferentes tipos de custos (materiais, mão de obra e serviços) de forma polimórfica.

A segunda fase trouxe desafios técnicos mais complexos, nomeadamente a implementação de uma arquitetura em camadas (N-Tier). A separação clara entre a lógica de negócio (Regras), o acesso aos dados (Dados) e as entidades (BO) exigiu um planeamento rigoroso, mas resultou num código muito mais organizado e profissional. A implementação da persistência de dados através de serialização e o tratamento de erros com exceções personalizadas (FicheiroException) foram passos cruciais para transformar um conjunto de classes numa aplicação robusta e funcional.

Repositório GitHub: https://github.com/tomascerqueiraa/ProjetoPOO.git

Bibliografia

- **Microsoft. (2022).** *Documentação Oficial do C#*. Microsoft Docs. Recuperado de: https://docs.microsoft.com/pt-br/dotnet/csharp/
- **W3Schools. (2021).** *Tutoriais de Programação C#*. Recuperado de: https://www.w3schools.com/cs/