PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)- LESI

RELATÓRIO DO TRABALHO DA DISCIPLINA DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Diogo Dias 26534

Diogo Gonçalves 26006

Resumo

Este trabalho da unidade curricular (UC) de Programação Orientada a Objetos (POO) consiste no desenvolvimento de um sistema que permite a gestão de uma loja online.

EST-IPCA – BARCELOS

novembro de 2023

Índice

Índice de Figuras

Introdução

No âmbito da unidade curricular de Programação Orientada a Objetos, lecionada pelo docente Luís Ferreira, foi proposto desenvolver um projeto em C# onde se pretendia soluções para problemas reais de complexidade moderada.

Optou-se pelo tema sugerido pelo docente, o comércio eletrónico, onde será implementado um sistema que permite a gestão de uma loja online.

Objetivos

Este trabalho tem, para além do inerente interesse académico, um forte interesse prático, destacando-se diversos objetivos essenciais. Em primeiro lugar, visa consolidar os conceitos fundamentais do Paradigma Orientado a Objetos, proporcionando uma compreensão sólida e aplicável deste modelo de programação.

Além disso, pretende-se realizar uma análise de problemas reais, permitindo a aplicação prática dos conhecimentos adquiridos e estimulando a resolução de desafios do mundo real.

A incorporação da linguagem de programação C# como parte integrante do trabalho visa desenvolver as competências de programação neste ambiente específico.

Além dos benefícios práticos, o projeto também visa potenciar a experiência no desenvolvimento de software. Isso implica não apenas a aplicação de técnicas específicas, mas também o desenvolvimento de habilidades colaborativas, gestão de projetos e resolução eficiente de problemas.

Por fim, o trabalho procura assegurar a assimilação completa do conteúdo da Unidade Curricular, garantindo não só a compreensão dos conceitos teóricos, mas também a capacidade de os aplicar de maneira efetiva em cenários práticos

Conceitos Importantes

Herança

A herança é um conceito fundamental na programação orientada a objetos que permite criar uma nova classe baseada em uma classe existente, aproveitando as propriedades e comportamentos da classe original. A classe recém-criada é chamada de classe derivada ou subclasse. A herança promove a reutilização de código, evitando a duplicação de implementações e facilitando a manutenção. Além disso, ela suporta a criação de hierarquias de classes, onde as classes mais específicas herdam características mais gerais.

Abstração

A abstração é um conceito que envolve a simplificação e representação de objetos do mundo real em um nível mais elevado de generalização. É utilizada em conjunto com a herança, permitindo a reutilização de código, diferindo desta, pois a classe derivada tem que obrigatoriamente reescrever os métodos abstratos da classe base.

Encapsulamento

Encapsulamento é o princípio de agrupar os dados (atributos) e os métodos (comportamentos) relacionados a um objeto em uma única classe. O encapsulamento restringe o acesso direto aos detalhes internos de um objeto, permitindo que o seu estado seja modificado apenas por meio de métodos cuidadosamente controlados (públicos). Isso ajuda a proteger a integridade dos dados e a ocultar a complexidade interna, facilitando a manutenção do código.

Polimorfismo

Polimorfismo refere-se à capacidade de um objeto se comportar de diferentes maneiras com base no contexto. O polimorfismo permite que diferentes métodos tenham o mesmo nome, mas aceitem parâmetros diferentes. O polimorfismo aumenta a flexibilidade e extensibilidade do código, permitindo que diferentes partes do sistema interajam de maneira consistente e adaptável.

Interfaces

Uma interface em programação define um conjunto de métodos que uma classe deve implementar. Ela proporciona um meio de definir comportamentos sem especificar a implementação real. Ao fornecer um conjunto de métodos que as classes devem implementar, as interfaces facilitam implementações e permitem a escrita de código mais flexível e extensível.

Exceções

Exceções são eventos paranormais ou situações de erro que ocorrem durante a execução de um programa. Elas são usadas para lidar com situações imprevistas e podem ser lançadas explicitamente pelo código ou surgir naturalmente devido a falhas durante a execução. O uso de exceções permite melhorias na legibilidade e manutenção do código. Utilizar exceções de uma maneira correta ajuda a criar um software mais robusto e a lidar com situações inesperadas de maneira controlada.

Verificações

Verificações referem-se à prática de validar dados ou condições antes de executar uma ação específica. Isso é essencial para garantir que o software funcione corretamente e de maneira segura. As verificações são frequentemente utilizadas para evitar erros lógicos ou de execução, melhorando a qualidade e a fiabilidade do código.

Projetos de Testes

Projetos de testes são conjuntos organizados de casos de teste que visam verificar a robustez do software. Eles incluem testes unitários. Projetos de testes são cruciais para garantir que as alterações no código não introduzam novos bugs e para validar que o software atende aos requisitos definidos. Eles também desempenham um papel fundamental na identificação precoce de problemas, melhorando a eficiência do processo de desenvolvimento.

Organização em N-tier

A organização em N-tier (ou camadas) é uma arquitetura de software que divide o sistema em diferentes camadas lógicas e funcionais. As camadas típicas incluem o frontend(interface do utilizador), regras de negócio e acesso a dados. Essa abordagem promove a escalabilidade e facilita a manutenção do sistema. As camadas transversais podem incluir aspetos como segurança, verificações e gerenciamento de exceções, garantindo que essas preocupações estejam distribuídas de maneira eficiente por todo o sistema. As interligações entre as camadas devem ser cuidadosamente implementadas, permitindo que cada camada funcione de forma independente e seja substituível sem afetar as outras.

1ª Fase

A primeira fase do projeto concentrou-se na estruturação inicial, tendo sido implementadas diversas classes e definidos diversos métodos e atributos fundamentais. Durante esta etapa, foram criadas as bases conceituais do sistema, delineando as relações entre as classes e estabelecendo a arquitetura global do projeto.

Esta fase inicial proporcionou uma sólida fundação para o desenvolvimento subsequente, fornecendo uma estrutura na qual as futuras iterações foram construídas.

Diagrama de classes

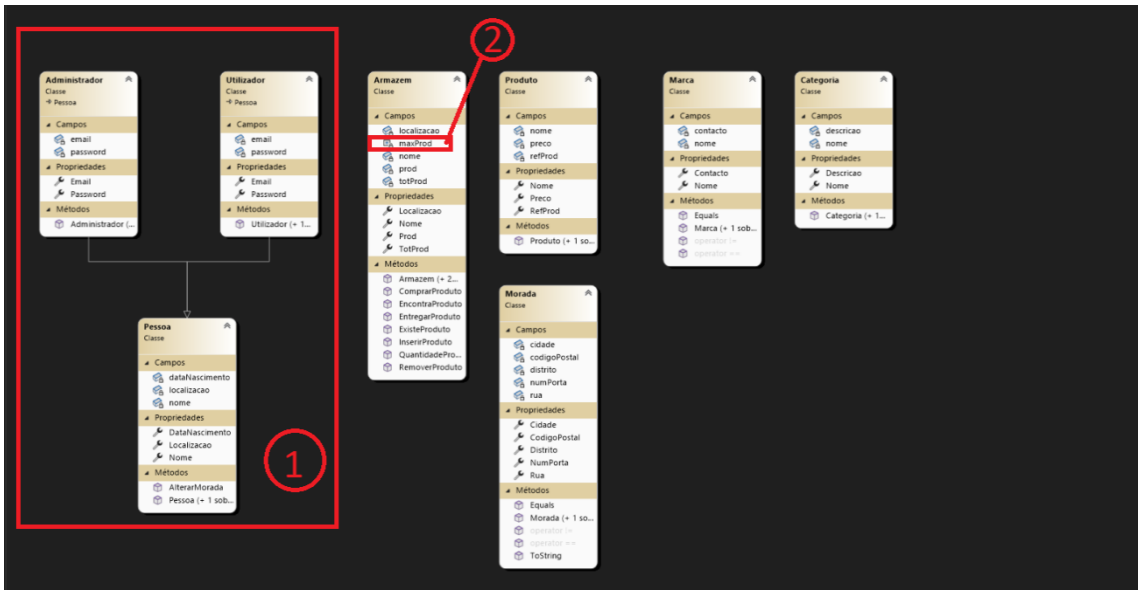

Figura 1 - Diagrama de classes do sistema desenvolvido na 1ª fase

Descrição diagrama de classes do sistema desenvolvido na 1ª fase

Neste sistema, a classe “Pessoa” contém propriedades comuns a todas as pessoas, como nome, data de nascimento e a respetiva morada. A classe “Utilizador” herda essas propriedades da classe “Pessoa” usando “*Utilizador : Pessoa*”, pois um utilizador também é considerado uma pessoa. Além das propriedades herdadas, ela possui duas propriedades específicas de utilizador, email e respetiva password. O mesmo acontece com a classe “Administrador”, como se pode ver no exemplo 1 da figura 1.

No que toca ao exemplo 2 da mesma figura, a razão pelo qual esse atributo

aparece com um ícone diferente é porque é uma constante, facilitando a mudança e reutilização da mesma em várias partes do código, caso necessário.

2ª Fase

Na segunda fase do projeto, a estratégia central foi direcionada para a otimização da organização e aprimoramento da qualidade da solução desenvolvida. A implementação de projetos de testes abrangentes desempenhou um papel crucial, validando a funcionalidade do sistema em várias dimensões, incluindo testes unitários. Essa abordagem sistemática permitiu uma cobertura extensa, identificando e corrigindo potenciais falhas de maneira proativa.

As verificações foram intensificadas para garantir a consistência dos dados e a integridade operacional do software. Isso incluiu a implementação de verificações de validação de dados abrangente e a incorporação de asserções para garantir que o código atenda às expectativas em diferentes cenários.

A gestão de exceções foi refinada para lidar eficientemente com situações inesperadas, proporcionando um tratamento adequado para cenários de erro e contribuindo para a resiliência global do sistema. As interfaces foram aprimoradas para maximizar a usabilidade e facilitar a interação do usuário, refletindo um compromisso contínuo com a experiência do utilizador final.

Um esforço substancial foi dedicado à documentação do código, fornecendo descrições detalhadas de classes, métodos e interfaces. A documentação não apenas serve como uma fonte valiosa de referência para programadores internos, mas também promove a transparência e facilita a integração de outros programadores.

Ao cumprir integralmente o plano de ação estabelecido para a segunda fase, foi alcançada não apenas a estabilidade técnica do sistema, mas também uma solução mais organizada, resiliente e de alta qualidade. O foco na organização, qualidade e documentação posicionou o projeto para uma evolução contínua e sustentável ao longo do desenvolvimento.

Descrição da organização de bibliotecas

De acordo com as orientações do docente Luís Ferreira, o projeto foi meticulosamente estruturado e organizado, refletindo um comprometimento inquestionável com os princípios lecionados.

A primeira DLL, intitulada "Dados", contém todas as classes plurais fundamentais do projeto.

A segunda DLL, denominada "Exceções", contém todas as classes de exceções personalizadas criadas.

A terceira DLL, "Frontend", contém uma classe “IO” onde se encontram todas as interações com o utilizador.

A DLL "ObjetosNegocio" constitui a quarta peça deste projeto, inlcuindo todas as classes singulares e interfaces presentes no projeto.

Finalmente, as DLLs "RegrasNegocio" e "Verificações" encerram a estrutura, representando o epicentro das operações, que em conjunto com as exceções, garantem a coesão e a validade do sistema. Na primeira, residem todas as regras que direcionam as classes plurais, enquanto na segunda, meticulosas verificações são realizadas para assegurar a integridade dos atributos em diversas instâncias.

Estas DLLs estão intricadamente conectadas, a biblioteca "Dados" mantém referências cruciais para "Exceções", "ObjetosNegocio" e "Verificacoes". Da mesma forma, "Frontend" interliga-se com "Exceções", "ObjetosNegocio", "RegrasNegocio" e, por último, "Verificações". Além disso, "RegrasNegocio" estabelece vínculos essenciais com "Dados", "Exceções", "ObjetosNegocio" e "Verificações", criando assim uma interdependência significativa entre todas as bibliotecas. Em suma, a abordagem organizacional adotada neste projeto atende todas as expectativas, proporcionando uma base sólida e eficiente para a materialização das ideias propostas.

Diagrama de classes

Figura 2 - Diagrama de classes do sistema desenvolvido na 2ª fase

Descrição diagrama de classes do sistema desenvolvido na 2ª fase

Como mencionado anteriormente, em linguagens de programação orientadas a objetos, como C#, as classes abstratas desempenham um papel importante na criação de hierarquias de classes.

No contexto deste sistema em C# e como é possível verificar no quadrado a vermelho maior na figura, a hierarquia de classes é organizada de forma que as classes Compra e Venda derivam da classe abstrata Operação. Essa abstração impõe a ambas a obrigação de implementar o método abstrato EmitirFatura, presente na classe base Operação. Essa abordagem promove a consistência no comportamento das operações, ao mesmo tempo em que permite que cada operação específica, seja ela de compra ou venda, forneça sua própria lógica personalizada para a emissão de faturas.

Utilizou-se também, a interface IComparable, como se visualiza na figura no quadrado vermelho menor, que é usada para permitir a comparação de objetos. Ela contém um único método chamado CompareTo, que compara o objeto atual com outro objeto do mesmo tipo. No contexto atual, são comparadas as datas das respetivas operações. Isto permite utilizar outros métodos predefinidos para Listas, por exemplo, como o lista.Sort().

Conclusão

O desenvolvimento deste projeto revelou-se um desafio complexo, no entanto, bastante enriquecedor. Durante todo o processo, desde a escolha do tema à realização do projeto, foram enfrentadas complexidades e obstáculos que desafiaram a expansão de conhecimentos e habilidades. Ao lidar com um sistema que permite a gestão de uma loja online, aprendeu-se e aprimorou-se capacidades de programação em C#, potenciou-se a experiência no desenvolvimento de software e assimilou-se todo o conteúdo lecionado na Unidade Curricular.

Embora o desafio tenha sido significativo, o resultado foi gratificante. Aprendeu-se a importância da colaboração em equipa, da comunicação eficaz e da capacidade de adaptação a situações em constante evolução. Este projeto forneceu uma base sólida para futuros empreendimentos e na área do desenvolvimento de software, trazendo certezas de que as lições aprendidas serão lembradas em futuros projetos, tanto a nível académico como a nível profissional.