

**Universidade Federal da Bahia**

**Disciplina**: MATA55 - Programação Orientada a Objetos

**Docente**: Profª Drª Rita Suzana Maciel

**Discentes**:Dimitri Santana Marinho (número de matrícula: 216215220)

Ravier de Oliveira Nascimento de Jesus (número de matrícula: 217125708)

**Contexto do Trabalho Final de POO**

**Objetivos**

Aplicar os conceitos aprendidos na disciplina de Programação Orientada a Objetos para desenvolver um software de e-commerce para uma Geek Store, isto é, uma loja que comercializa produtos relacionados à cultura nerd e ao universo de entretenimento

**Descrição e Justificativa**

O trabalho visa aplicar os conceitos de POO estudados em sala, tais como sobrecarga, sobreposição, composição, herança, polimorfismo, classes e métodos estáticos, ligação dinâmica, pacotes, classes abstratas, interfaces, entre outros conceitos presentes na ficha de avaliação do trabalho.

A escolha por desenvolver um software de e-commerce ocorreu pela familiaridade com o tema, devido ao desenvolvimento de exercícios semelhantes do Moodle, e também por ser possível pensar de forma clara e objetiva na aplicação de todos os conceitos estudados. A escolha da Geek Store como tema do e-commerce surgiu por sugestão de um dos membros, sendo acatado pelo interesse mútuo da dupla.

**Diagrama UML**

O programa é composto por 12 pacotes, desta forma, decidiu-se que os diagramas UML serão expressos em Anexo.

**Conceitos Aplicados e Motivação**

**Sobrecarga:** Pacote exibirClientes, Classe ExibirClientes

Utilizou-se a sobrecarga no método de exibirPessoa, utilizando a mesma assinatura, mudando o parâmetro para exibição de Pessoa Física ou de Pessoa Jurídica.

**Sobreposição:** Pacote: geekStore, Classes Abstratas e Suas Classes Herdeiras

Utilizou-se a sobreposição para alterar o método toString() das classes herdeiras.

**Composição:** Pacote geekStore, Classe Loja

Utilizou-se ArrayLists de Produto, Cliente, FormaPagamento e Funcionario, para aplicar métodos inerentes ao e-commerce, tais como adicionar/remover produtos/clientes/forma de pagamentos, etc.

**Herança:** Pacote: geekStore, Todos os Produtos da loja, exemplo na Classe Camisa

Utilizou-se Herança ao criar classes abstrata (Produtos, Títulos, Acessórios) e fazer com que Objetos a serem vendidos fossem herdados dessas classes.

**Polimorfismo:** Pacote exibirProdutos, Classe ExibirProdutos

Utilizou-se polimorfismo de sobrecarga, ver a explicação no item **Sobrecarga**

**Ligação Dinâmica:** Pacote cadastroProdutos, Classe CadastroProdutos

Utilizou-se a ligação dinâmica no cadastro dos produtos para ligar as características de um produto ao respectivo tipo que queríamos cadastrar

**Campos Estáticos:** Pacote Loja, Classe Loja. Todos os Atributos dessa Classe

Utilizou-se campos estáticos no pacote Loja para garantir que os campos fossem compartilhados por todas as instâncias da classe. E os métodos estáticos para que pudessem ser executados sem que as instâncias fossem criadas e que a alteração do valor por qualquer instância se propagasse para as outras instâncias da classe, com o intuito de implementar rotinas da loja que fossem independentes dos dados armazenados na classe.

**Instance of e Cast:** Pacote consultarProdutos, Classe ConsultarProdutosCliente

Utilizou-se o instanceof e o Cast para especificar qual produto está a ser consultado para chamar corretamente a exibição do menu com as informações do produto.

**Classes Abstratas e Metodos Abstratos:** Pacote geekStore. As Classes Produto, Acessório e Titulo são abstratas e o método abstrato na classe Produto

Utilizou-se as classes abstratas para melhor organização dos produtos (e seus atributos, no construtor) e o método abstrato toString() na classe abstrata Produto.

**Interfaces:** Pacote geekStore

Utilizou-se da interface AlterrarCarrinho para implementar o método adicionarItem para adição dos produtos ao carrinho de compras.

**Tratamento e Lançamento de Exceção:** Pacote cadastroProdutos, Classe CadastroProdutos

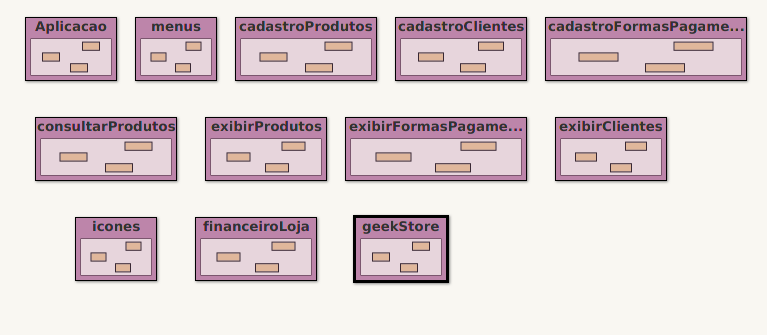
Utilizou-se o Tratamento e Lançamento de Exceção para garantir a correta inserção dos produtos no cadastro, não permitindo, por exemplo, a inserção de números negativos, entre outros casos de exceção.

**Interface Gráfica:** Pacote menus, Classe Menus

Utilizou-se o JOptionPane para implementar o menu com interface gráfica para o programa.

**ANEXO**

**Figura 1 - Pacotes existentes no trabalho**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 2 - Diagrama UML do Pacote Aplicacao**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 3 - Diagrama UML do Pacote menus**



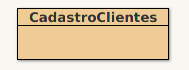
Fonte: Autoria própria (produzido no BlueJ)

**Figura 4 - Diagrama UML do Pacote cadastroProdutos**



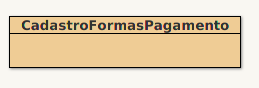
Fonte: Autoria própria (produzido no BlueJ)

**Figura 5 - Diagrama UML do Pacote cadastroClientes**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 6 - Diagrama UML do Pacote cadastroFormasPagamento**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 7 - Diagrama UML do Pacote consultarProdutos**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 8 - Diagrama UML do Pacote exibirProdutos**



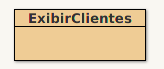
Fonte: Autoria própria (produzido no BlueJ)

**Figura 9 - Diagrama UML do Pacote exibirFormasPagamento**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 10 - Diagrama UML do Pacote exibirClientes**



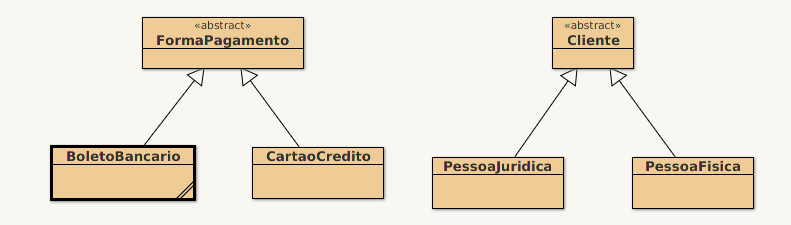
Fonte: Autoria própria (produzido no BlueJ)

**Figura 11 - Diagrama UML do Pacote icones**



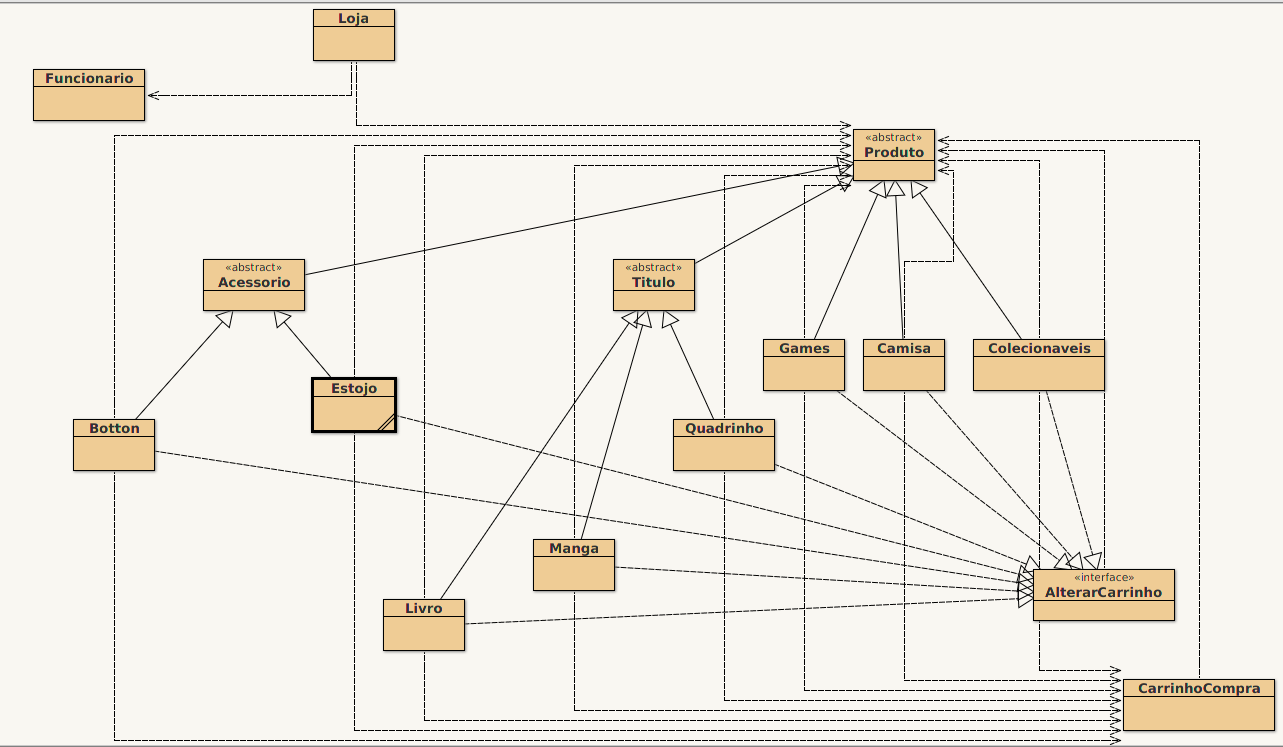
Fonte: Autoria própria (produzido no BlueJ)

**Figura 12 - Diagrama UML do Pacote financeiroLoja**



Fonte: Autoria própria (produzido no BlueJ)

**Figura 13 - Diagrama UML do Pacote geekStore**



Fonte: Autoria própria (produzido no BlueJ)