**UNIVERSIDADE FEDERAL DO RIO GRANDE (FURG)**

MATHEUS TEIXEIRA DE SOUZA

Relatório Sobre um Projeto em [UML](https://ava.furg.br/mod/resource/view.php?id=514136) e Implementação em Smalltalk Utilizando a Linguagem Pythom e Orientação a Objetos

Rio Grande - RS

2023

Relatório sobre projeto utilizando programação orientada a objetos (POO) em Python

Matheus Teixeira de Souza1

1Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)

matheusts@furg.br

**1. Introdução**

Nesse relatório será mostrado um projeto em UML implementado na linguagem Python utilizando o suporte a orientação a objetos da linguagem, foi programado um suporte para o controle de estoques utilizando as devidas classes fornecidas.

Após foram feitas chamadas das devidas classes e funções para testar se as classes foram implementadas adequadamente.

**2. Estrutura do Código:**

O código está estruturado em classes, cada uma representando uma entidade específica. As principais classes são:

* **Produto:** Representa um produto em estoque, com atributos como código, nome, quantidade, unidade e uma lista de preços associados. Possui métodos para adicionar preços, imprimir informações do produto e verificar preços.
* **Preco:** Representa o preço associado a um produto em um determinado período. Contém lógicas para alocar um item ao preço, imprimir informações sobre o preço e localizar um preço com base em uma data.
* **Venda:** Representa uma venda, contendo informações como código, data, hora e uma lista de itens de venda associados. Possui métodos para adicionar itens, imprimir informações da venda e remover itens.
* **ItensVenda:** Representa um item em uma venda, contendo informações sobre a data, o produto associado, a quantidade e a venda à qual pertence. Contém lógicas para alocar uma venda, imprimir informações do item e remover uma venda.
* **Estoque:** Representa o estoque global da empresa, contendo listas de produtos e vendas. Possui métodos para adicionar, consultar, imprimir e remover produtos e vendas do estoque.
* **Funções Auxiliares:** Foram implementadas funções para realizar operações específicas, como adicionar um novo produto ao estoque, adicionar uma nova venda, consultar produtos e vendas, imprimir informações, inicializar o estoque e remover produtos e vendas.

**3. Analise detalhada das funções e classes:**

**3.1 Classes:**

**Classe Produto:**

* **Atributos:**
  + **codigo**: Identificador único do produto.
  + **nome**: Nome do produto.
  + **quantidade**: Quantidade disponível em estoque.
  + **unidade**: Unidade de medida do produto.
  + **listaprecos**: Lista de preços associados ao produto.
* **Métodos:**
  + **adicionaPreco(preco)**: Adiciona um novo preço à lista de preços do produto.
  + **devolucao(quantidade)**: Processa a devolução de uma quantidade específica do produto.
  + **executaBaixa(quantidade)**: Realiza a baixa de produtos do estoque.
  + **imprimeItem**: Imprime informações básicas sobre o produto.
  + **inicializa()**: Inicializa o produto (lógica de inicialização).
  + **imprimeProduto()**: Imprime informações detalhadas sobre o produto, incluindo preços associados.
  + **removePreco(preco)**: Remove um preço específico da lista de preços.
  + **retornaCodigo()**: Retorna o código do produto.
  + **verificaPreco()**: Verifica o preço atual do produto.
  + **verificaProduto(codigo)**: Verifica se o produto tem um código específico.

**Classe Preco:**

* **Atributos:**
  + **datainicial**: Data de início de validade do preço.
  + **datafinal**: Data de término de validade do preço.
  + **item**: Item associado ao preço.
  + **preco**: Valor do preço.
* **Métodos:**
  + **alocaItem()**: Aloca um item ao preço, considerando a validade.
  + **imprimePreco()**: Imprime informações sobre o preço.
  + **inicializa()**: Inicializa o preço (lógica de inicialização).
  + **localizaPreco(data)**: Localiza o preço válido com base em uma data específica.

**Classe Venda:**

* **Atributos:**
  + **codigo**: Identificador único da venda.
  + **data**: Data da venda.
  + **hora**: Hora da venda.
  + **itensVenda**: Lista de itens associados à venda.
* **Métodos:**
  + **adicionaItem(item)**: Adiciona um item à lista de itens da venda.
  + **inicializa()**: Inicializa a venda (lógica de inicialização).
  + **imprimeVenda()**: Imprime informações detalhadas sobre a venda.
  + **removeItemVenda(item)**: Remove um item específico da lista de itens.

**Classe** **ItensVenda:**

* **Atributos:**
  + **data**: Data do item de venda.
  + **item**: Item associado ao item de venda.
  + **quantidade**: Quantidade do item na venda.
  + **venda**: Venda associada ao item.
* **Métodos:**
  + **alocaVenda()**: Aloca a venda associada ao item.
  + **devolveProduto()**: Processa a devolução do produto associado ao item.
  + **inicializa()**: Inicializa o item (lógica de inicialização).
  + **localiza()**: Localiza informações sobre o item de venda.
  + **imprimeItemVenda()**: Imprime informações detalhadas sobre o item de venda.
  + **removeVenda()**: Remove a venda associada ao item.

**Classe** **Estoque:**

* **Atributos:**
  + **empresa**: Nome da empresa.
  + **listadeprodutos**: Lista de produtos em estoque.
  + **listadevendas**: Lista de vendas registradas.
* **Métodos:**
  + **adicionaProduto(produto)**: Adiciona um novo produto ao estoque.
  + **adicionaVenda(venda)**: Adiciona uma nova venda ao estoque.
  + **consultaProduto(codigo)**: Consulta um produto no estoque com base no código.
  + **consultaVenda(codigo)**: Consulta uma venda no estoque com base no código.
  + **imprimeProdutos()**: Imprime informações sobre todos os produtos no estoque.
  + **imprimeVendas()**: Imprime informações sobre todas as vendas no estoque.
  + **inicializa()**: Inicializa o estoque (lógica de inicialização).
  + **removeProduto(codigo)**: Remove um produto do estoque com base no código.
  + **removeVenda(codigo)**: Remove uma venda do estoque com base no código.

**3. Funcionalidades Principais:**

* **Adição e Remoção de Produtos e Vendas:** O sistema permite a adição e remoção de produtos e vendas ao estoque, proporcionando flexibilidade na gestão de inventário.
* **Consulta e Impressão de Informações:** Funções foram implementadas para consultar e imprimir informações detalhadas sobre produtos e vendas no estoque.
* **Associação de Preços aos Produtos:** A classe Produto permite a associação de preços a um produto, facilitando a gestão de diferentes valores ao longo do tempo.

**4. Exemplo de Uso:**

O código foi testado com um exemplo de uso, onde foram criados um estoque, um produto, um preço associado a esse produto, uma venda e um item de venda. Operações como adição, consulta, impressão e remoção foram realizadas para demonstrar o funcionamento do sistema.

**5. Considerações Finais:**

O desenvolvimento deste código de controle de estoques proporcionou uma compreensão mais aprofundada de vários conceitos fundamentais de programação orientada a objetos (POO) e estruturas de dados. As aprendizagens principais foram:

**5.1 Abstração e Modelagem:**

O processo de modelagem de entidades (como produtos, preços, vendas) em classes permite uma representação mais fiel e estruturada dos componentes do sistema.

**5.2 Encapsulamento e Modularidade:**

O uso de classes e métodos permite encapsular o comportamento e os dados relacionados a cada entidade, promovendo a modularidade do código. Isso facilita a manutenção e extensão do sistema.

**5.3 Herança e Polimorfismo:**

Embora não tenhamos explorado herança e polimorfismo de forma extensiva neste código, a estrutura do código fornece uma base que pode ser estendida para incorporar esses conceitos, especialmente quando há comportamentos comuns entre diferentes entidades.

**5.4 Uso de Listas e Estruturas de Dados:**

A implementação de listas para armazenar produtos, preços e vendas destaca a importância do uso de estruturas de dados eficientes para manipulação e organização de informações.

**5.5 Lógica de Negócios e Integração:**

O desenvolvimento deste código envolveu a implementação de lógicas de negócios específicas para controle de estoques, como adição, remoção, consulta e operações relacionadas a produtos e vendas. Isso destaca a importância de compreender as regras de negócios ao projetar sistemas.

**5.6 Manipulação de Datas:**

O uso da biblioteca datetime para manipulação de datas ressalta a importância de considerar a temporalidade em sistemas que lidam com eventos ao longo do tempo.

**5.7 Testes e Exemplos Práticos:**

O desenvolvimento de exemplos práticos, como a criação de um estoque, produtos, preços e vendas, proporcionou uma oportunidade para testar as funcionalidades do código, garantindo que as lógicas implementadas estejam corretas.

Em resumo, o desenvolvimento deste código proporcionou uma aplicação prática dos conceitos fundamentais de POO, estruturas de dados e lógicas de negócios. A compreensão desses conceitos é crucial para o desenvolvimento de sistemas mais robustos, modulares e adaptáveis. Além disso, a prática de documentação e revisão crítica é essencial para a qualidade e a manutenção do código ao longo do tempo.