Bruno Ricardo de Sá Ferreira

2201529 - Número de estudante

Engenharia Informática - curso

Linguagens de Programação - Disciplina

Desenvolvi este projeto em etapas, começando pela função principal oferecida pelo professor, que serviu como base para a implementação das funcionalidades requeridas. Durante o processo, fiz ajustes na forma de leitura dos dados da base de dados para garantir compatibilidade, sem comprometer a lógica do programa.

As principais funções implementadas foram as seguintes:

1. **Funções de Cálculo(**Ocaml**):**

    - calculate_total_price: Calcula o preço total dos itens no carrinho sem descontos.
    - calculate_category_discounts: Calcula os descontos por categoria para os itens no carrinho.
    - calculate_loyalty_discount: Calcula o desconto de lealdade com base nos anos de fidelidade do cliente.
    - calculate_shipping_cost: Obtém o custo de envio com base no distrito de entrega.
    - calculate_final_price: Calcula o preço final do carrinho após aplicar todos os descontos e adicionar o custo de envio.

2. **Funções de Exibição:**

    - display_cart: Exibe os itens do carrinho com suas respectivas categorias, quantidades e preços.

3. **Função** main**:**

    - Coordena a execução do programa, realiza a leitura do arquivo store.pl e o parsing das informações.
    - Exibe os descontos por categoria, itens disponíveis, descontos de lealdade e custos de envio.
    - Calcula o preço total do carrinho, descontos por categoria, desconto de lealdade, custo de envio e preço final do carrinho.
    - Exibe os resultados dos cálculos e os itens do carrinho.

Relatório Detalhado da Classe OperacoesCarrinho(Java)**:**

1. **Introdução:**
    - Responsável por coordenar operações relacionadas ao carrinho de compras, como cálculos de preços, descontos e custos de envio e utilizar subprocessos para executar código OCaml, mantendo o programa simples e evitando repetição de lógica, esta é considerado a função mais importante e desafiadora do projeto.

2. **Métodos de Execução OCaml:**
    - Possui um método privado executarOCaml para executar comandos OCaml e retornar a saída como uma lista de strings. Esse método utiliza subprocessos para compilar e executar o código OCaml.
    - Cada método de cálculo chama executarOCaml com os parâmetros apropriados e interpreta a saída para obter os resultados desejados.
3. **Cálculos de Preço e Desconto:**
    - Função responsável por passar a lógica de calculo implementada em Ocaml. Esta não é responsável por cálculos complexos mas sim por fazer a "transição" de um código para o outro.
4. **Cálculo do Preço Final:**
    - calcularPrecoFinal combina os resultados dos cálculos anteriores para determinar o preço final do carrinho. Recebe os valores do preço total, desconto por categoria, desconto de lealdade e custo de envio como parâmetros.
5. **Exibição do Carrinho de Compras:**
    - exibirCarrinho exibe o carrinho de compras utilizando o código OCaml correspondente. Retorna uma string contendo os itens do carrinho formatados.
6. **Processamento de Resultados:**
    - Cada método que chama o código OCaml interpreta a saída para extrair os resultados desejados. Por exemplo, os valores de desconto são extraídos a partir das linhas que começam com a informação relevante.
    - A saída é processada linha por linha, permitindo que os resultados sejam recuperados de forma eficiente e precisa.

1. **Método** main.Java()**:**

    - Criamos uma nova instância da classe Store para representar a loja.
    - Criamos um novo cliente com algumas informações fictícias, como ID, cidade, distrito e anos de lealdade.
    - Adicionamos itens fictícios ao carrinho do cliente, utilizando a classe Item.
    - Adicionamos o cliente à loja.
2. **Cálculos e Exibição:**
    - Calculamos o preço total do carrinho, o desconto por categoria, o desconto de lealdade e o custo de envio usando métodos da classe OperacoesCarrinho.
    - Com base nesses cálculos, determinamos o preço final do carrinho.
    - Exibimos o carrinho de compras, o desconto de lealdade, o custo de envio, o desconto por categoria, o preço total sem descontos e o preço final do carrinho na saída padrão.
3. **Decisões Lógicas:**
    - Utilizamos objetos das classes Cliente, Item, Store e métodos da classe OperacoesCarrinho para modularizar e organizar o código.

- Optamos por utilizar valores fictícios para clientes e itens, facilitando a demonstração do funcionamento do programa.
- Decidimos calcular o preço final do carrinho considerando descontos por categoria, desconto de lealdade e custo de envio, proporcionando uma simulação mais realista do processo de compra.
- Utilizamos a classe OperacoesCarrinho para capturar os cálculos complexos, mantendo o código principal mais legível e organizado.

Após a execução dos testes, observamos que o programa produziu resultados consistentes e precisos.

1. **Descontos por Categoria:**
   - Os descontos para cada categoria foram corretamente calculados e exibidos, com valores de 10%, 20%, 30%, 15% e 25% para as categorias "potions", "wands", "enchanted_books", "crystals" e "amulets", respectivamente.
2. **Desconto de Lealdade:**
   - Os descontos de lealdade foram aplicados com precisão, variando de 5% a 25% com base nos anos de fidelidade do cliente.
3. **Custo de Envio:**
   - O custo de envio para cada distrito foi calculado corretamente, com valores de 5.00, 7.00 e 10.00 para os distritos "Aveiro", "Lisboa" e "Porto", respectivamente.
4. **Preço Total sem Descontos:**
   - O preço total do carrinho sem descontos foi calculado corretamente como 110.00.
5. **Preço Final do Carrinho:**
   - O preço final do carrinho, após aplicação de todos os descontos e custo de envio, foi calculado como 75.00.
6. **Itens no Carrinho de Compras:**
   - Os itens no carrinho de compras foram exibidos corretamente, com informações detalhadas sobre cada item, incluindo categoria, quantidade e preço.

Esses testes foram realizados com cinco produtos diferentes, permitindo verificar a precisão dos cálculos e a lógica implementada no programa Em suma, o programa foi concebido com a lógica mais simples possível, visando facilitar o entendimento e a manutenção do código. No entanto, ao lidar com cálculos complexos e integração com código OCaml, os subprocessos tornaram-se uma peça essencial para concluir o projeto com sucesso. Estes desempenharam um papel crucial na comunicação entre o código Java e o código OCaml. Eles permitiram que os resultados gerados pelo código OCaml fossem passados de volta para o código Java, possibilitando a integração eficiente de ambas as partes do sistema.

-