Trabalho de POO

Nomes: Arthur Donadussi e Pedro Kuntz
Professor: Jaqson Dalbosco

Projeto: Cafeteria
Linguagem: C++

Software: Cafeteria

Objetivo: O software visa gerenciar pedidos em uma cafeteria, permitindo o cadastro de clientes, produtos, e manipulação de pedidos com funcionalidades como adicionar e remover produtos, aplicar descontos, e calcular o total do pedido.

Diagrama de Classes:

Produto: Classe base para produtos com atributos como nome e preço.

Comida: Derivada de Produto, adiciona calorias.

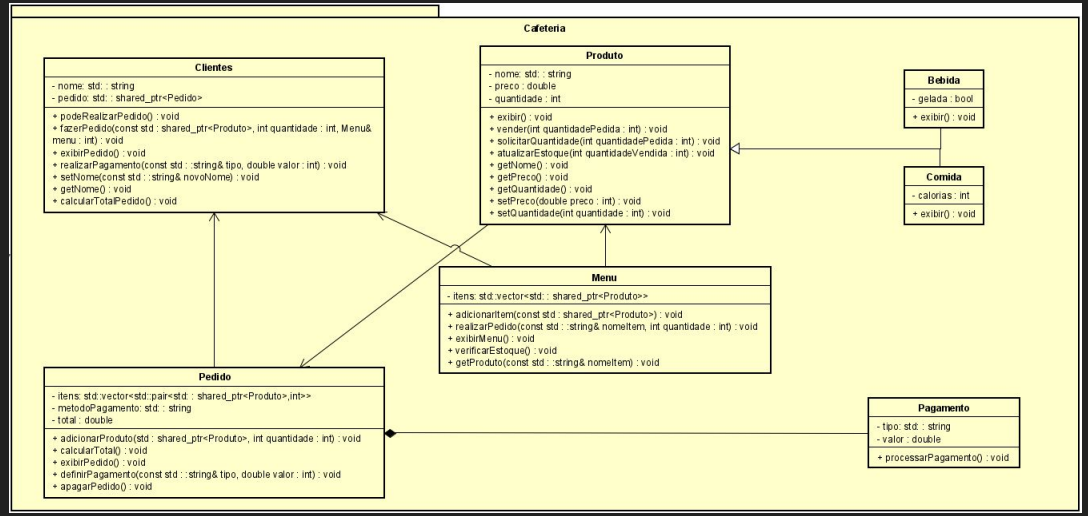
Bebida: Derivada de Produto, adiciona se está gelada.

Pedido: Gerencia uma lista de produtos, permite adicionar, remover produtos, calcular total e aplicar descontos.

Cliente: Associa um nome a um pedido, gerencia contato do cliente, e permite salvar e carregar pedidos.

Menu: Gerencia uma lista de produtos disponíveis. Permite adicionar itens ao menu e realizar pedidos verificando a disponibilidade dos produtos.

Pagamento: Associa um tipo de pagamento e um valor. Permite processar o pagamento.



Descrição das Classes:

Produto: Métodos para exibir informações, acessar e modificar o preço.

Comida e Bebida: Métodos específicos para exibir características únicas como calorias e se a bebida está gelada.

Pedido: Gerenciamento de produtos através de métodos para adicionar, e calcular o total.

Cliente: Gestão de pedidos e informações do cliente. Salvamento e carregamento do estado do pedido.

Menu: Gerencia uma lista de produtos disponíveis, facilitando a realização de pedidos.

Pagamento: Responsável por processar os pagamentos associados aos pedidos, armazenando informações sobre o tipo de pagamento e o valor.

Cliente.cpp

Cliente.cpp > fazerPedido(const std::shared_ptr<Produto>&, int, Menu &)

```cpp
#include "Cliente.h"
#include <iostream>

Cliente::Cliente() : pedido(std::make_shared<Pedido>()) {}

bool Cliente::podeRealizarPedido() const {
    if (nome.empty()) {
        std::cerr << "Erro: Nome do cliente não pode ser vazio para realizar pedidos." << std::endl;
        return false;
    }
    return true;
}

void Cliente::fazerPedido(const std::shared_ptr<Produto>& produto, int quantidade, Menu& menu) {
    if (!podeRealizarPedido()) return;
    if (produto->getQuantidade() >= quantidade) {
        produto->vender(quantidade);
        pedido->adicionarProduto(produto, quantidade);
    } else {
        std::cerr << "Quantidade insuficiente para o produto: " << produto->getNome() << std::endl;
    }
}

void Cliente::exibirPedido() const {
    pedido->exibirPedido();
}

void Cliente::realizarPagamento(const std::string& tipo, double valor) {
    if (!podeRealizarPedido()) return;
    pedido->definirPagamento(tipo, valor);
}

void Cliente::setNome(const std::string& novoNome) {
    if (!novoNome.empty()) {
        nome = novoNome;
        std::cout << "Nome configurado: " << nome << std::endl;
    } else {
        std::cerr << "Erro: Nome não pode ser vazio." << std::endl;
    }
}

std::string Cliente::getNome() const {
    return nome;
}

double Cliente::calcularTotalPedido() const {
    return pedido->calcularTotal();
}
```

Linguagem de Programação Escolha: C++

Criador: Bjarne Stroustrup, no início dos anos 1980, como uma extensão da linguagem C para suportar programação orientada a objetos e outras melhorias.

Por que C++? C++ oferece controle detalhado de gerenciamento de memória e suporta polimorfismo, o que é crucial para uma estrutura de classes complexa e robusta. Esta linguagem permite a construção de sistemas onde o controle de baixo nível é necessário, juntamente com a flexibilidade de um alto nível de abstração.

Vantagem: C++ é conhecida por sua eficiência e velocidade de execução, o que é vital para aplicações que requerem otimização de recursos.

Desvantagem: A flexibilidade e o nível de controle que C++ oferece vêm com uma curva de aprendizado mais acentuada em comparação a linguagens de mais alto nível.

C++: Arquivos .h

Utilizar arquivos de cabeçalho (.h ou .hpp) em C++ não é obrigatório, mas é uma prática recomendada por várias razões:

1. **Modularidade e Organização**: Arquivos de cabeçalho ajudam a organizar o código, separando as declarações das definições. Isso facilita a leitura e manutenção do código, especialmente em projetos grandes.
2. **Reutilização de Código**: Declarações de classes, funções e variáveis podem ser incluídas em múltiplos arquivos fonte (.cpp) através de #include, promovendo a reutilização do código.
3. **Compilação Independente**: Mudanças em um arquivo de implementação (.cpp) não exigem a recompilação de outros arquivos que dependem dele, desde que o cabeçalho não tenha sido alterado. Isso pode economizar tempo durante a compilação de grandes projetos.
4. **Separação de Interface e Implementação**: Cabeçalhos permitem que a interface de uma classe (suas funções públicas e variáveis) seja separada da sua implementação (o código real). Isso é útil para fornecer bibliotecas onde os usuários só precisam saber o que uma classe pode fazer, não como ela faz.

Objetivos do Trabalho:

1. Classes de Domínio Relacionadas:

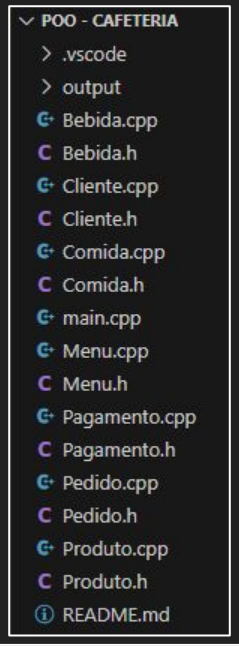
- Comida, Bebida, Cliente, Menu, Pedido, Produto, Pagamento
- Essas classes estão inter-relacionadas, com Comida e Bebida herdados de Produto, e Cliente e Menu que manipulam objetos de Pedido e Produto.

2. Herança e Outro Tipo de Relacionamento:

- **Herança**: Comida e Bebida são subclasses de Produto.
- **Associação**: Cliente possui um objeto Pedido e interage com Menu e Produto.
- **Agregação**: Menu contém uma lista de produtos (Comida e Bebida).

3. Métodos com Regras de Negócio:

- Menu::realizarPedido(): Verifica estoque e realiza pedidos.
- Produto::vender(): Vende produto e atualiza estoque.
- Pedido::definirPagamento(): Define o método e o valor do pagamento após calcular o total do pedido.




```cpp
#include "Comida.h"

Comida::Comida(const std::string& nome, double preco, int calorias, int quantidadeInicial)
    : Produto(nome, preco, quantidadeInicial), calorias(calorias) {}
```

```cpp
bool Menu::realizarPedido(const std::string& nomeItem, int quantidade) {
    for (auto& item : itens) {
        if (item->getNome() == nomeItem) {
            std::cout << "Quantidade inicial do produto " << nomeItem << ": " << item->getQuantidade() << std::endl;
            if (item->getQuantidade() >= quantidade) {
                return true;
            } else {
                std::cerr << "Quantidade insuficiente para o produto: " << nomeItem << std::endl;
                return false;
            }
        }
    }
    std::cerr << "Produto não encontrado: " << nomeItem << std::endl;
    return false;
}
```

Objetivos do Trabalho:

4. Método de Inicialização, Construtores:

- Todos os objetos principais como Comida, Bebida, Cliente, Menu, Pedido são inicializados corretamente em seus construtores.

5. Exemplo de Sobrecarga e/ou Substituição:

- **Substituição**: O método exibir() é sobrescrito em Comida e Bebida para incluir detalhes específicos além dos apresentados na classe Produto.

6. Fluxo de Execução para Exemplificar Instanciação e Mensagens Entre Objetos:

- O main.cpp ilustra a instanciação de produtos, adição ao menu, criação de um cliente, pedidos sendo feitos, e o pagamento sendo realizado. Mostra claramente a interação entre as classes e como os objetos interagem e mantêm estado.

```cpp
Bebida::Bebida(const std::string& nome, double preco, bool gelada, int quantidadeInicial)
    : Produto(nome, preco, quantidadeInicial), gelada(gelada) {}
```

```cpp
void Bebida::exibir() const {
    Produto::exibir();
    std::cout << "Gelada: " << (gelada ? "Sim" : "Nao") << std::endl;
}
```

```cpp
void Comida::exibir() const {
    Produto::exibir();
    std::cout << "Calorias: " << calorias << std::endl;
}
```

```cpp
int main() {
    std::string nomeCliente;
    int escolha = 0;

    // Boas-vindas
    std::cout << "Bem-vindo a Cafeteria!" << std::endl;
    std::cout << "Por favor, digite seu nome: ";
    getline(std::cin, nomeCliente);

    // Instanciando objetos necessários
    Cliente cliente;
    cliente.setNome(nomeCliente);
    Menu menu;

    // Adiciona produtos ao menu
    menu.adicionarItem(std::make_shared<Bebida>("Cafe", 2.50, false, 10)); // Gelada: false
    menu.adicionarItem(std::make_shared<Comida>("Bolo", 3.00, 250, 5)); // Nome atualizado para "Bolo"
    menu.adicionarItem(std::make_shared<Bebida>("Refri", 2.00, true, 20)); // Novo item "Refri"
    menu.adicionarItem(std::make_shared<Comida>("Pao de Queijo", 1.50, 100, 15)); // Novo item "Pão de Queijo"
```

Objetivos do Trabalho:

7. Implementação do fluxo de execução em console para manipulação de objetos em memória

- O usuário possui opções para escolher, dentre elas o menu, a disponibilidade de cada item, fazer um pedido, ver o seu pedido, realizar o pagamento do pedido e sair da aplicação

```
do {
    std::cout << "\nO que voce gostaria de fazer?" << std::endl;
    std::cout << "1. Mostrar Menu" << std::endl;
    std::cout << "2. Verificar disponibilidade de estoque" << std::endl;
    std::cout << "3. Fazer um pedido" << std::endl;
    std::cout << "4. Ver pedido" << std::endl;
    std::cout << "5. Realizar pagamento" << std::endl;
    std::cout << "6. Sair" << std::endl;
    std::cout << "Escolha uma opcao: ";
    std::cin >> escolha;

    std::cin.ignore();  // Ignora o '\n' restante

    switch (escolha) {
        case 1:
            menu.exibirMenu();
            break;
        case 2:
            menu.verificarEstoque();
            break;
        case 3:
            {
                std::string nomeProduto;
                int quantidade;
                std::cout << "Digite o nome do produto: ";
                getline(std::cin, nomeProduto);
                std::cout << "Digite a quantidade desejada: ";
                std::cin >> quantidade;
                std::cin.ignore();  // Limpa o buffer
                auto produto = menu.getProduto(nomeProduto);
                if (produto) {
                    cliente.fazerPedido(produto, quantidade, menu);
                    std::cout << "Pedido realizado com sucesso!" << std::endl;
                } else {
                    std::cout << "Nao foi possivel realizar o pedido. Produto nao disponivel ou quantidade insuficiente." << std::endl;
                }
            }
            break;
        case 4:
            std::cout << "Pedido do Cliente: " << cliente.getNome() << std::endl;
            cliente.exibirPedido();
            break;
        case 5:
            {
                double valorPagamento = cliente.calcularTotalPedido();
                std::string tipoPagamento;
                std::cout << "Digite o tipo de pagamento (Dinheiro, Cartao, Pix): ";
                getline(std::cin, tipoPagamento);
                cliente.realizarPagamento(tipoPagamento, valorPagamento);
                if (tipoPagamento == "Dinheiro" || tipoPagamento == "Cartao" || tipoPagamento == "Pix") {
                    // Processar pagamento
                    Pagamento pagamento(tipoPagamento, valorPagamento);
                    pagamento.processarPagamento();
                    std::cout << "Pagamento realizado com sucesso. Valor: $" << valorPagamento << std::endl;
                }
            }
            break;
        case 6:
            std::cout << "Obrigado por usar a Cafeteria. Ate logo!" << std::endl;
            break;
        default:
            std::cout << "Opção invalida. Tente novamente." << std::endl;
    }
} while (escolha != 6);
```

Conclusão:

Neste trabalho, utilizando a linguagem C++ para fazer um sistema de gerenciamento de pedidos para uma cafeteria. Implementamos diversas funcionalidades essenciais para o funcionamento de um restaurante, como o cadastro de produtos, a realização e exibição de pedidos, além do processamento de pagamentos.

A escolha da linguagem C++ se mostrou vantajosa para este projeto, proporcionando um controle detalhado do gerenciamento de memória e suporte a polimorfismo, aspectos cruciais para a construção de um sistema complexo e eficiente.

Em resumo, este projeto não só atingiu os objetivos propostos, como também destacou a capacidade do C++ em gerenciar sistemas de pedidos de maneira eficiente e organizada, comprovando sua relevância e eficácia em cenários de programação orientada a objetos.

VALEU PELAS AULAS JAQSON!