**Aula de Typesrcipt 11/10/2024 professor Gustavo**

**Classe Peça**

* **id:** Número único para identificação da peça.
* **nome:** Nome da peça (e.g., "Camiseta", "Calça").
* **descricao:** Descrição detalhada da peça.
* **tamanho:** Tamanho da peça (e.g., "P", "M", "G").
* **cor:** Cor da peça.
* **preco:** Preço da peça.
* **quantidade:** Quantidade em estoque.

**Classe Cliente**

* **id:** Número único para identificação do cliente.
* **nome:** Nome completo do cliente.
* **email:** Endereço de email.
* **telefone:** Número de telefone.
* **endereco:** Endereço completo.

**Classe Venda**

* **id:** Número único para identificação da venda.
* **cliente:** Objeto da classe Cliente.
* **data:** Data da venda.
* **itens:** Array de objetos, cada um com a peça e a quantidade vendida.
* **valorTotal:** Valor total da venda.

**Implementando as Classes em TypeScript**

TypeScript

class Peca {

constructor(

public id: number,

public nome: string,

public descricao: string,

public tamanho: string,

public cor: string,

public preco: number,

public quantidade: number

) {}

}

class Cliente {

constructor(

public id: number,

public nome: string,

public email: string,

public telefone: string,

public endereco: string

) {}

}

class Venda {

constructor(

public id: number,

public cliente: Cliente,

public data: Date,

public itens: { peça: Peca, quantidade: number }[],

public valorTotal: number

) {

this.valorTotal = this.calcularValorTotal();

}

private calcularValorTotal(): number {

return this.itens.reduce((total, item) => total + item.peça.preco \* item.quantidade, 0);

}

}

Use o código [com cuidado](file:///C:\faq#coding).

**Funcionalidades Adicionais**

* **CRUD:** Criação, leitura, atualização e deleção de peças, clientes e vendas.
* **Gerenciamento de Estoque:** Atualizar a quantidade em estoque ao realizar uma venda.
* **Relatórios:** Gerar relatórios de vendas por período, cliente, etc.
* **Pesquisa:** Pesquisar peças por nome, tamanho, cor, etc.
* **Cadastro de Clientes:** Cadastrar novos clientes e consultar informações existentes.
* **Finalização de Vendas:** Criar novas vendas, adicionando itens ao carrinho e calculando o valor total.
* **Pagamento:** Simular diferentes formas de pagamento (cartão, dinheiro, etc.).
* **Emissão de Nota Fiscal:** Gerar uma nota fiscal para cada venda.

**Considerações Adicionais**

* **Banco de dados:** Para armazenar os dados de forma persistente, você pode utilizar um banco de dados relacional (MySQL, PostgreSQL) ou não relacional (MongoDB).
* **Interface do usuário:** Para interagir com o sistema, você pode criar uma interface web utilizando frameworks como React, Angular ou Vue.js.
* **Validação de dados:** Implementar validações para garantir a integridade dos dados, como verificar se a quantidade em estoque é suficiente para uma venda.
* **Segurança:** Implementar medidas de segurança para proteger os dados dos clientes, como criptografia de senhas.

**Próximos Passos**

* **Design da interface:** Definir a estrutura e o layout da interface do usuário.
* **Implementação da persistência:** Escolher um banco de dados e implementar as operações de CRUD.
* **Lógica de negócio:** Implementar as funcionalidades de cadastro, venda, relatório, etc.
* **Testes:** Criar testes unitários para garantir a qualidade do código.

**Este é apenas um ponto de partida.** O desenvolvimento de um sistema completo de gerenciamento de loja de roupas envolve diversas outras considerações e pode se tornar um projeto complexo.

**Classe Sessão**

* **id:** Identificador único da sessão.
* **filme:** Título do filme.
* **sala:** Número da sala.
* **horario:** Horário de início da sessão.
* **assentos:** Um array bidimensional representando os assentos da sala (true para ocupado, false para livre).

**Classe Reserva**

* **id:** Identificador único da reserva.
* **cliente:** Nome do cliente.
* **sessao:** Objeto da classe Sessão.
* **assentos:** Um array dos assentos reservados.

**Classe Relatório**

* **periodo:** Período do relatório (dia, semana, mês).
* **totalVendas:** Valor total das vendas no período.
* **filmesMaisAssistidos:** Lista dos filmes mais assistidos no período.
* **salasMaisUtilizadas:** Lista das salas mais utilizadas no período.

**Implementando as Classes em TypeScript**

TypeScript

class Sessao {

constructor(

public id: number,

public filme: string,

public sala: number,

public horario: Date,

public assentos: boolean[][]

) {}

}

class Reserva {

constructor(

public id: number,

public cliente: string,

public sessao: Sessao,

public assentos: number[][]

) {}

}

class Relatorio {

constructor(

public periodo: string,

public totalVendas: number,

public filmesMaisAssistidos: { filme: string, quantidade: number }[],

public salasMaisUtilizadas: { sala: number, quantidade: number }[]

) {}

}

Use o código [com cuidado](file:///C:\faq#coding).

**Funcionalidades Adicionais**

* **Cadastro de Sessões:** Cadastrar novas sessões, incluindo informações sobre o filme, sala e horário.
* **Reserva de Assentos:** Permitir que os clientes reservem assentos para uma determinada sessão.
* **Geração de Relatórios:** Gerar relatórios de vendas, ocupação das salas e filmes mais assistidos.
* **Cancelamento de Reservas:** Permitir que os clientes cancelem suas reservas.
* **Gerenciamento de Salas:** Definir a capacidade de cada sala e os tipos de assentos (normal, VIP, etc).
* **Integração com Pagamento:** Implementar a integração com um gateway de pagamento para processar as transações.
* **Notificações:** Enviar notificações por email ou SMS para os clientes sobre suas reservas e alterações.

**Considerações Adicionais**

* **Banco de dados:** Utilizar um banco de dados para armazenar as informações sobre sessões, reservas e relatórios.
* **Interface do usuário:** Criar uma interface intuitiva para que os usuários possam realizar reservas e consultar informações.
* **Validação de dados:** Implementar validações para garantir que os dados inseridos sejam consistentes e válidos.
* **Escalabilidade:** Projetar o sistema para que possa lidar com um grande número de sessões e reservas.

**Classe Paciente**

* **id:** Identificador único do paciente.
* **nome:** Nome completo do paciente.
* **dataNascimento:** Data de nascimento.
* **CPF:** Número do CPF.
* **convenio:** Convênio médico.
* **historicoMedico:** Um array de objetos representando o histórico médico do paciente (data, descrição, diagnóstico, etc.).

**Classe Médico**

* **id:** Identificador único do médico.
* **nome:** Nome completo do médico.
* **CRM:** Número do CRM.
* **especialidade:** Especialidade médica.

**Classe Internação**

* **id:** Identificador único da internação.
* **paciente:** Objeto da classe Paciente.
* **medico:** Objeto da classe Médico.
* **dataEntrada:** Data de entrada do paciente.
* **dataSaida:** Data de saída do paciente (pode ser null se o paciente ainda estiver internado).
* **diagnostico:** Diagnóstico principal.
* **procedimentos:** Um array de objetos representando os procedimentos realizados (data, descrição, valor).

**Classe Estoque**

* **id:** Identificador único do item em estoque.
* **nome:** Nome do item.
* **quantidade:** Quantidade em estoque.
* **tipo:** Tipo do item (medicamento, material hospitalar, etc.).

**Funcionalidades Adicionais**

* **Cadastro de Pacientes, Médicos e Itens de Estoque:** Cadastrar novos registros e atualizar informações existentes.
* **Internação de Pacientes:** Registrar novas internações, associando o paciente a um médico e um leito.
* **Alta de Pacientes:** Registrar a alta de um paciente, atualizando a data de saída da internação.
* **Gerenciamento de Estoque:** Controlar a entrada e saída de itens do estoque.
* **Agendamento de Consultas:** Agendar consultas entre médicos e pacientes.
* **Emissão de Relatórios:** Gerar relatórios de internações, consultas, estoque, etc.
* **Gerenciamento de Leitos:** Controlar a ocupação dos leitos.

**Considerações Adicionais**

* **Banco de dados:** Utilizar um banco de dados relacional (MySQL, PostgreSQL) ou não relacional (MongoDB) para armazenar os dados de forma persistente.
* **Interface do usuário:** Criar uma interface intuitiva para que os usuários possam realizar as diversas operações do sistema.
* **Validação de dados:** Implementar validações para garantir a integridade dos dados, como verificar se o CPF do paciente é válido.
* **Segurança:** Implementar medidas de segurança para proteger os dados dos pacientes e garantir a confidencialidade das informações médicas.
* **Escalabilidade:** Projetar o sistema para que possa lidar com um grande volume de dados e usuários.

**Próximos Passos**

* **Design da interface:** Definir a estrutura e o layout da interface do usuário.
* **Implementação da persistência:** Escolher um banco de dados e implementar as operações de CRUD.
* **Lógica de negócio:** Implementar as funcionalidades de cadastro, internação, alta, agendamento, etc.
* **Testes:** Criar testes unitários para garantir a qualidade do código.

**Classes Propostas:**

* **Morador:**
  + Nome
  + Apartamento
  + Telefone
  + E-mail
  + Contatos de emergência
  + Histórico de pagamentos
  + Permissões de acesso
* **Aluguel:**
  + Imóvel
  + Inquilino (referência para a classe Morador)
  + Valor
  + Data de início
  + Data de vencimento
  + Status (pago, pendente, cancelado)
* **Manutenção:**
  + Descrição
  + Data de solicitação
  + Data de execução
  + Custo
  + Status (pendente, em andamento, concluída)
  + Responsável
* **Serviço:**
  + Tipo (limpeza, segurança, etc.)
  + Frequência
  + Contratado (empresa ou pessoa física)
  + Custo
* **Relatório:**
  + Tipo (financeiro, manutenção, etc.)
  + Período
  + Dados gerados
* **Ferramenta:**
  + Nome
  + Descrição
  + Funcionalidade
* **Segurança:**
  + Módulo para controle de acesso
  + Monitoramento de câmeras
  + Alarmes
* **Gestão de Pagamentos:**
  + Emissão de boletos
  + Registro de pagamentos
  + Conciliação bancária

## Considerações Adicionais

* **Relações entre classes:** Utilize interfaces para definir as relações entre as classes, como por exemplo, um aluguel ter um inquilino (classe Morador).
* **Herança:** Utilize herança para criar classes mais específicas, como diferentes tipos de manutenção ou serviços.
* **Interfaces:** Defina interfaces para representar dados que serão compartilhados entre diferentes partes do sistema, como por exemplo, um objeto de configuração.
* **Módulos:** Organize o código em módulos para facilitar a manutenção e a reutilização.
* **Banco de dados:** Utilize um banco de dados relacional ou NoSQL para armazenar os dados do sistema. TypeScript oferece integração com diversos bancos de dados através de ORMs ou drivers específicos.
* **Framework:** Considere utilizar um framework como Angular ou React para construir a interface do usuário.
* **Testes:** Escreva testes unitários para garantir a qualidade do código.

## Próximos Passos

* **Detalhamento das classes:** Definir todos os atributos e métodos de cada classe.
* **Relações entre classes:** Estabelecer as relações entre as classes através de interfaces e associações.
* **Design da interface do usuário:** Criar um design intuitivo e fácil de usar para o sistema.
* **Implementação:** Desenvolver o código das classes e funções.
* **Integração com banco de dados:** Conectar o sistema a um banco de dados e implementar as operações de leitura e escrita.
* **Testes:** Escrever testes unitários para garantir a qualidade do código.