📦 Contexto Geral: Sistema de Reservas para Restaurantes

A plataforma **MesaJá** é um sistema simples de gestão de reservas que conecta **clientes** e **restaurantes**.
O objetivo do desafio é simular, em memória, a lógica de criação e gerenciamento de reservas de mesas considerando:

- **Preferência de local** (salão ou jardim)
- **Período do dia** (manhã, tarde ou noite)
- **Capacidade máxima de cada mesa**
- **Conflito de reservas** (uma mesa não pode ser reservada por dois clientes no mesmo período)

✅ Não é necessário criar interface gráfica ou integração com banco de dados.
✅ Toda a solução deve funcionar com classes, métodos e estruturas de dados em memória (List/Map/Set).

Premissas e Regras Gerais

- O **CPF** identifica um cliente no sistema (não é obrigatório validar formato; trate como String).
- Dentro de um restaurante, o **número da mesa** é o identificador único de uma mesa.
- Uma reserva deve ser criada informando **quantidade de pessoas**, **preferência de local** e **período**.
- O restaurante pode confirmar reservas, manter uma lista de espera e gerar relatórios.

🧩 Etapa 1 – Modelagem de Classes Básicas (POO)

1. Descrição / Objetivo

O objetivo desta etapa é estruturar os principais componentes do sistema de reservas utilizando conceitos de **Programação Orientada a Objetos (POO)**. Você deverá modelar as entidades fundamentais do domínio (Cliente, Restaurante, Mesa e Reserva), além dos enums que descrevem local, período e status.

Como este desafio não utiliza banco de dados, todas as informações existirão apenas em memória. Por isso, é importante que suas classes fiquem bem definidas e prontas para serem utilizadas nas etapas seguintes, onde serão implementadas as regras de disponibilidade, criação/cancelamento de reservas e geração de relatórios.

Ao final desta etapa, o sistema deve estar preparado para:

- Criar clientes com seus dados básicos (nome, CPF, endereço e telefone)
- Criar restaurantes contendo sua identificação (nome e endereço), além de listas internas para mesas e reservas
- Criar mesas com número, capacidade máxima e local (salão/jardim)
- Criar reservas contendo os dados da solicitação, mesmo que ainda não seja possível confirmá-las nesta etapa

---

2. Entidades e atributos obrigatórios

🧑 Cliente

Representa a pessoa que deseja realizar uma reserva.

- **nome** (String)
- **cpf** (String)

Observação: o CPF será utilizado como identificador do cliente no sistema. Não é necessário validar formato neste desafio (trate como texto).

🏪 Restaurante

Representa o estabelecimento onde as reservas serão realizadas.
O restaurante será responsável por armazenar suas mesas e reservas em memória.

- **nome** (String)
- **endereco** (String)
- **mesas** (List)
- **reservas** (List) *(reservas registradas no restaurante; ao longo do desafio, você poderá separar "confirmadas" e "lista de espera" em propriedades separadas conforme necessário)*

🍽️ Mesa

Representa uma mesa física dentro do restaurante.

- **numero** (int)
- **capacidade** (int) *(quantidade máxima de pessoas suportada pela mesa)*
- **local** (LocalMesa)

Observação: o número da mesa é o identificador único dentro de um restaurante (não pode existir duas mesas com o mesmo número no mesmo restaurante).

📌 Reserva

Representa uma solicitação de reserva feita por um cliente.

- **cliente** (Cliente)
- **restaurante** (Restaurante)

- **mesa** (Mesa) *(pode iniciar como null e ser atribuída apenas quando a reserva for confirmada nas etapas futuras)*
- **quantidadePessoas** (int)
- **localPreferencia** (LocalMesa)
- **periodo** (PeriodoReserva)
- **status** (StatusReserva) (deve ser preenchido conforme o sistema identificar a possibilidade de confirmar a reserva)

---

3. Enumeradores

Recomendação: Caso não possua conhecimento de Enumeradores, apenas trate-os como propriedades **String,** simplificando a implementação, mas se atenha aos valores que serão utilizados nessas propriedades.

LocalMesa

Indica a área do restaurante onde a mesa está localizada e/ou a preferência do cliente.

- SALAO
- JARDIM

PeriodoReserva

Indica o período do dia para o qual a reserva é solicitada.

- MANHA
- TARDE
- NOITE

StatusReserva

Indica o estado atual de uma reserva.

- PENDENTE
- CONFIRMADA
- CANCELADA

---

4. Entregáveis da Etapa 1

Ao concluir esta etapa, você deve ter:

- **Todas as classes e enums** criados conforme o enunciado
- Construtores funcionais para permitir instanciar objetos e iniciar testes simples no console
- Estrutura pronta para evoluir nas próximas etapas (principalmente Restaurante e Reserva)

🧩 Etapa 2 – Operações sobre o Restaurante (Gestão de Mesas e Busca)

1. Descrição / Objetivo

O objetivo desta etapa é adicionar funcionalidades ao restaurante que permitam **gerenciar suas mesas** e realizar uma primeira busca de disponibilidade de forma simples e objetiva.

Como não estamos utilizando banco de dados, todas as mesas serão mantidas em memória, e o **número da mesa** deve ser tratado como identificador único dentro do restaurante.
O foco principal será garantir que não existam mesas duplicadas e que as mesas cadastradas tenham informações válidas (principalmente a capacidade).

Ao final desta etapa, o restaurante deve ser capaz de:

- Adicionar mesas com validações básicas
- Impedir cadastro de mesas com número duplicado

- Buscar mesas compatíveis com uma solicitação, considerando capacidade mínima e local desejado

2. Descrição Técnica

Método adicionarMesa(Mesa mesa) — (Implementação na Classe Restaurante)

- Deve validar se a mesa possui **capacidade maior que zero**
- Deve impedir a inclusão de uma mesa cujo **número já exista** no restaurante
- Deve exibir uma mensagem apropriada no console para sucesso ou falha

Assinatura sugerida:

```java
public void adicionarMesa(Mesa mesa) ...

public void adicionarMesa(params) ...
```

Observação: caso o construtor do Restaurante receba uma lista/array de mesas, ele deve aplicar as mesmas validações ao inicializar a lista interna.

Método buscarDisponibilidade(int quantidadePessoas, LocalMesa local) — (Implementação na Classe Restaurante)

Este método deve retornar uma **lista contendo as mesas que atendem** aos seguintes critérios:

- **Capacidade da mesa maior ou igual** à quantidade de pessoas solicitada
- **Localização da mesa** igual ao local informado (salão ou jardim)

Assinatura sugerida:

```
Java
public List<Mesa> buscarDisponibilidade(int quantidadePessoas,
LocalMesa local)
```

Importante: nesta etapa, o método **não precisa considerar reservas existentes**. Essa validação será adicionada na Etapa 4.

🧩 Etapa 3 – Criação de Reservas (Confirmação Inicial)

1. Descrição / Objetivo

O objetivo desta etapa é implementar a lógica de criação de reservas no restaurante.
A partir de uma solicitação feita por um cliente, o sistema deverá avaliar se existe uma mesa compatível com a **quantidade de pessoas**, o **local preferido** e o **período desejado**.

Caso seja possível atender à solicitação, a reserva deve ser criada com status **CONFIRMADA** e registrada no restaurante.
Caso não exista nenhuma mesa compatível (por capacidade e/ou local), o sistema deve exibir uma mensagem clara informando que não foi possível confirmar a reserva.

Ao final desta etapa, o restaurante deve ser capaz de:

- Criar uma reserva a partir de uma solicitação de cliente
- Selecionar e atribuir uma mesa compatível (quando existir)
- Registrar reservas confirmadas em sua lista interna

2. Descrição Técnica

📌 Método criarReserva(...) — (Implementação na Classe Restaurante)

Parâmetros mínimos:

- Cliente
- Restaurante
- Quantidade de pessoas
- Local de preferência (salão ou jardim)
- Período (manhã, tarde, noite)

Regras:

- O sistema deve buscar uma mesa com **capacidade suficiente** e no **local solicitado**
- Se existir mesa compatível:
    - Criar a reserva atribuindo a mesa escolhida ao objeto
    - Definir status como **CONFIRMADA**
    - Armazenar a reserva nas reservas do restaurante
- Se não existir mesa compatível:
    - Não criar reserva confirmada
    - Exibir uma mensagem de indisponibilidade no console

Assinatura sugerida:

```java
public Reserva criarReserva(Cliente cliente, int quantidadePessoas,
LocalMesa localPreferido, PeriodoReserva periodo)
```

O **ideal** é que o sistema selecione a mesa **mais adequada (menor capacidade viável)**, e não apenas com capacidade suficiente. Exemplo: Se há 2 mesas disponíveis no restaurante, a primeira para 7 pessoas, e a segunda para 5, e é feita uma solicitação de reserva para 4 pessoas, o ideal é que o sistema selecione a mesa para **5 pessoas** visto que é a **menor capacidade viável**.

🧩 Etapa 4 – Validação de Conflitos (Mesa ocupada no período)

1. Descrição / Objetivo

O objetivo desta etapa é tornar o sistema mais próximo de um cenário real, impedindo que o restaurante confirme reservas conflitantes.
A partir de agora, não basta existir uma mesa com capacidade e local adequados: o sistema também deve verificar se a mesa já está reservada naquele período.

Em outras palavras: **uma mesa não pode ter duas reservas confirmadas para o mesmo período**.

Ao final desta etapa, o restaurante deve ser capaz de:

- Avaliar disponibilidade real de uma mesa, considerando reservas já confirmadas
- Impedir conflito de reservas para a mesma mesa no mesmo período
- Continuar confirmando reservas normalmente quando existir mesa livre

2. Descrição Técnica

- O método criarReserva(...) deve verificar as reservas já registradas no restaurante.
- Uma mesa é considerada indisponível no período caso já exista reserva **CONFIRMADA** para:
    - a mesma mesa
    - o mesmo período
- Reservas **CANCELADAS** não devem bloquear uma mesa (cheque apenas status "**CONFIRMADA**".

Dica: Utilize iterações na coleção de reservas confirmadas do restaurante para verificar se há alguma reserva para a mesa assim que o sistema

encontrar uma mesa “viável” (capacidade e local de acordo com a solicitação).

🧩 Etapa 5 – Lista de Espera e Promoção Automática após Cancelamento

1. Descrição / Objetivo

O objetivo desta etapa é adicionar um comportamento comum em restaurantes: quando não é possível confirmar uma reserva, o cliente pode optar por entrar em uma **lista de espera**.

Caso um cliente aceite entrar na lista de espera, sua solicitação deve ser registrada como uma reserva com status **PENDENTE**.
Posteriormente, se uma reserva confirmada for cancelada e uma mesa for liberada naquele período/local, o sistema deve tentar aproveitar essa liberação para **promover automaticamente** uma reserva pendente para o status **CONFIRMADA**.

Ao final desta etapa, o restaurante deve ser capaz de:

- Registrar solicitações pendentes em uma lista de espera
- Cancelar reservas confirmadas
- Promover automaticamente uma reserva pendente compatível quando houver liberação

2. Descrição Técnica

✅ Evolução do método `criarReserva(...)`

Novo parâmetro:

- aceitaListaEspera (boolean)

Regras adicionais:

- Se não existir mesa disponível:
    - Se aceitaListaEspera for verdadeiro:
        - Registrar a reserva com status **PENDENTE**
        - Armazenar a reserva na lista de espera do restaurante - você pode utilizar uma lista separada para controlar a lista de espera, separadamente da lista de reservas, facilitando o entendimento
    - Caso contrário:
        - Exibir mensagem informando que não foi possível confirmar a reserva

Assinatura sugerida:

```java
public Reserva criarReserva(
    Cliente cliente,
    int quantidadePessoas,
    LocalMesa localPreferido,
    PeriodoReserva periodo,
    boolean aceitaListaEspera
)
```

❌ Funcionalidade de cancelamento

A reserva poderá ser cancelada, alterando o status para **CANCELADA**.
Ao cancelar uma reserva confirmada, o restaurante deve:

1. Identificar que uma mesa ficou disponível naquele período

2. Procurar na lista de espera uma reserva pendente compatível (mesmo período, mesmo local e capacidade atendida)
3. Promover essa reserva para **CONFIRMADA**, atribuindo uma mesa e movendo-a para a lista de reservas confirmadas

Assinatura sugerida (duas opções — vocês escolhem uma):

```java
public void cancelarReserva(Reserva reserva)
```

ou

```java
public void cancelarReserva(String idReserva)
```

🧩 Etapa 6 – Relatórios (CPF → Reservas Confirmadas)

1. Descrição / Objetivo

O objetivo desta etapa é avaliar a capacidade de trabalhar com **estruturas de mapeamento (Map/HashMap/Dictionary)** para construir um relatório consolidado.

Dado um conjunto de CPFs e um conjunto de restaurantes, o sistema deverá mapear cada CPF para todas as reservas **CONFIRMADAS** encontradas nesses restaurantes.
A saída deve ser organizada e fácil de conferir no console, indicando quantas reservas cada cliente possui.

Ao final desta etapa, o sistema deve ser capaz de:

- Consolidar reservas confirmadas de múltiplos restaurantes
- Agrupar reservas por CPF usando um Map
- Exibir um resumo claro do resultado

2. Descrição Técnica

Assinatura sugerida:

```java
public static Map<String, List<Reserva>>
gerarRelatorioReservasConfirmadasPorCpf(
    List<String> cpfs,
    List<Restaurante> restaurantes
)
```

Regras mínimas:

- Considerar apenas reservas com status **CONFIRMADA**
- O relatório deve incluir apenas os CPFs fornecidos como entrada (não listar CPFs “descobertos” no sistema)
- **Exibir no console um resumo por CPF (ex.: CPF → total de reservas confirmadas)**