Exercício: Desenvolvimento de um Sistema de Mensagens semelhante ao Discord

Introdução:

Você foi contratado para desenvolver um sistema básico de mensagens em Java, semelhante ao Discord. O objetivo é aplicar os conceitos dos quatro pilares da Programação Orientada a Objetos (abstração, encapsulamento, herança e polimorfismo) e utilizar estruturas de controle (if, for, while). Além disso, na segunda parte do exercício, você irá aprimorar o sistema utilizando a **Stream API** do Java.

Parte 1: Implementação Básica com Conceitos de POO

Objetivo:

Desenvolver um conjunto de classes que representem as entidades do sistema de mensagens, atendendo aos requisitos funcionais descritos abaixo. Você deverá **identificar as classes necessárias**, definir seus atributos e métodos, e implementar interações entre elas usando os conceitos de OO.

Requisitos Funcionais:

1. **Gerenciamento de Usuários**:

• O sistema deve permitir a criação de usuários com um nome de usuário e um ID único.

• Cada usuário pode participar de múltiplos servidores (canais).

2. **Servidores (Canais)**:

• O sistema deve permitir a criação de servidores (canais) onde os usuários podem enviar mensagens.

• Cada servidor possui um nome e uma lista de usuários participantes.

• Deve ser possível adicionar e remover usuários de um servidor.

3. **Mensagens**:

• Usuários podem enviar mensagens em um servidor.

• Cada mensagem deve conter o conteúdo, o autor (usuário) e a data/hora de envio.

• O sistema deve manter um histórico de mensagens em cada servidor.

4. **Interações Básicas**:

• Usuários podem visualizar as mensagens de um servidor.

• Deve ser possível listar todos os usuários de um servidor.

• Implementar funcionalidades para simular uma conversa em um servidor.

5. **Requisitos Técnicos**:

• Utilizar os **quatro pilares da Programação Orientada a Objetos**:

• **Abstração**: Modelar entidades reais do sistema.

• **Encapsulamento**: Proteger os dados das classes e fornecer acesso através de métodos.

• **Herança**: Identificar relações "é um" entre classes e reutilizar código.

• **Polimorfismo**: Permitir que objetos de diferentes classes sejam tratados de forma unificada.

• Utilizar **estruturas de controle** (if, for, while) onde apropriado.

Instruções:

1. **Identificação das Classes**:

• Analise os requisitos e identifique as classes necessárias (por exemplo, Usuario, Servidor, Mensagem).

• Defina as relações entre as classes.

2. **Definição dos Atributos e Métodos**:

• Para cada classe, defina os atributos necessários.

• Implemente métodos que realizem as ações descritas nos requisitos.

3. **Implementação**:

• Desenvolva as classes em Java, aplicando os conceitos de POO.

• Utilize estruturas de controle para implementar a lógica do sistema.

4. **Programa Principal**:

• Crie uma classe com o método main para testar o sistema.

• Simule a criação de usuários, servidores e o envio de mensagens.

5. **Comentários e Documentação**:

• Comente o código explicando cada parte.

• Explique como os conceitos de OO foram aplicados.

Exemplo de Fluxo do Sistema:

• Crie usuários “Alice” e “Bob”.

• Crie um servidor chamado “Programação”.

• Adicione “Alice” e “Bob” ao servidor “Programação”.

• “Alice” envia uma mensagem: “Olá, pessoal!”.

• “Bob” envia uma mensagem: “Olá, Alice!”.

• Liste as mensagens do servidor “Programação”.

Parte 2: Aprimoramento com Stream API

Objetivo:

Implementar novas funcionalidades no sistema que permitam manipular e consultar dados de forma eficiente utilizando a **Stream API** do Java.

Requisitos Funcionais Adicionais:

1. **Consulta Avançada de Mensagens**:

• Permitir a filtragem de mensagens por autor ou por palavras-chave.

• Exibir as mensagens filtradas de forma ordenada (por data/hora ou autor).

2. **Estatísticas dos Servidores**:

• Calcular o número total de mensagens em um servidor.

• Encontrar o usuário que mais enviou mensagens em um servidor.

• Listar todos os usuários que enviaram mensagens contendo uma determinada palavra.

3. **Agrupamento de Mensagens**:

• Agrupar mensagens por autor.

• Exibir a contagem de mensagens por usuário.

Instruções:

1. **Atualização das Classes Existentes**:

• Adicione métodos que retornem coleções necessárias para o uso da Stream API.

2. **Implementação das Funcionalidades**:

• Utilize a Stream API para implementar os novos requisitos.

3. **Programa Principal**:

• No método main, demonstre o uso das novas funcionalidades.

4. **Comentários e Documentação**:

• Comente o código explicando como a Stream API está sendo utilizada.

Dicas Gerais:

• **Abstração**: Pense nas entidades principais (usuários, servidores, mensagens) e como representá-las.

• **Encapsulamento**: Proteja os dados das classes com modificadores de acesso adequados.

• **Herança e Polimorfismo**: Considere se alguma classe pode herdar de outra ou se há interfaces que podem ser implementadas.

• **Stream API**: Utilize operações como filter(), map(), collect(), groupingBy(), etc.