Trabalho em Dupla: Simulador de Fila de Atendimento

Visão Geral

Este trabalho tem como objetivo a implementação de um simulador de fila de atendimento em Java, utilizando os conceitos de Programação Orientada a Objetos (POO) e Estruturas de Dados abordados em sala de aula, com foco especial em Filas encadeada(Queue) e Pilhas (Stack). Os alunos deverão desenvolver um sistema que simule o fluxo de clientes em um ambiente de atendimento, como um banco, hospital ou loja, gerenciando diferentes tipos de filas e prioridades. É fundamental que a solução demonstre a compreensão e aplicação prática das estruturas de dados de forma eficiente e organizada.

Objetivos de Aprendizagem

Ao final deste trabalho, os alunos deverão ser capazes de:

- Aplicar os princípios da Programação Orientada a Objetos (classes, objetos, agregação) na modelagem de entidades do mundo real.
- Implementar e manipular estruturas de dados lineares como Filas (Queue) e Pilhas (Stack) de forma eficiente.
- Compreender e aplicar o conceito de filas de prioridade.
- Desenvolver algoritmos para gerenciar o fluxo de atendimento e calcular estatísticas básicas.
- Organizar o código de forma modular, legível e bem documentada.
- Trabalhar em equipe, dividindo tarefas e integrando módulos de código.

Regras

- Este é um trabalho em dupla. Os nomes de ambos os participantes devem constar no topo da classe Main, como um comentário no código, e também no arquivo README.md, localizado na pasta principal.
- O prazo de entrega é final e improrrogável. Preste atenção ao horário.
- Em caso de trabalho copiado, a nota será dividida entre os envolvidos.
- O código que não for executado não poderá ser corrigido, resultando em nota zero.
- A correção será realizada no IntelliJ IDEA 2025.1.3 (Community Edition). Caso o código não seja executado nesta plataforma, ele não poderá ser avaliado.

Cenário Sugerido: Atendimento Bancário

Para contextualizar o simulador, sugere-se o cenário de um banco com diferentes tipos de atendimento e filas:

- **Atendimento Comum:** Clientes para operações gerais (saques, depósitos, etc.). Fila FIFO padrão.
- **Atendimento Prioritário:** Clientes idosos, gestantes, pessoas com deficiência. Fila de prioridade.

O cenário pode ser adaptado para um hospital (emergência, consultas, exames) ou uma loja (caixas, devoluções, SAC), desde que mantenham a complexidade e os requisitos de estruturas de dados similares.

Classes Principais Sugeridas

Para a implementação, as seguintes classes são sugeridas como ponto de partida. Os alunos podem adicionar outras classes conforme a necessidade do projeto:

`Cliente`

Representa um cliente que chega ao banco para ser atendido.

- **Atributos:**
  - `String nome`: Nome do cliente.
  - `String tipoAtendimento`: Tipo de atendimento desejado (ex: "Comum", "Prioritário").
  - `int prioridade`: Nível de prioridade (ex: 1 para prioritário, 0 para comum). Pode ser inferido do `tipoAtendimento`.
  - `long tempoChegada`: Timestamp da chegada do cliente (para cálculo de tempo de espera).
  - `String atendente`: Nome da pessoa que fez o atendimento.
- **Métodos:**
  - Getters para todos os atributos.
  - Outros que o grupo achar necessário.

`SistemaAtendimento`

Classe principal que orquestra o simulador, gerenciando as filas e os atendentes.

- **Atributos:**
  - `FilaEncadeada filaComum`: Fila para clientes comuns.
  - `FilaEncadeada filaPrioritaria`: Fila para clientes prioritários.
  - `PilhaEncadeada historicoAtendimentos`: Pilha para registrar o histórico de atendimentos (ex: "Cliente X atendido por Atendente Y"). Esta pilha será fundamental para a funcionalidade de "desfazer" operações, conforme detalhado nas funcionalidades mínimas requeridas.
- **Métodos:**
  - `adicionarCliente(Cliente cliente)`: Adiciona um cliente à fila apropriada.
  - `chamarProximoCliente()`: Lógica para verificar qual atendente está livre e qual cliente deve ser chamado da fila (prioridade, FIFO).
  - `gerarRelatorio()`: Gera um relatório com estatísticas.
  - `desfazerUltimoAtendimento()`: Utiliza a pilha de histórico para simular um "desfazer".

Conceitos e Estruturas de Dados a Serem Aplicados

Este trabalho exige a aplicação dos seguintes conceitos e estruturas de dados:

- **Programação Orientada a Objetos (POO):** Classes, objetos, encapsulamento (modificadores de acesso `public`/`private`), construtores, métodos.
- **Filas (Queue):** Utilização da implementação manual de uma fila para a fila comum e de prioridade.
- **Pilhas (Stack):** Utilização da implementação manual de uma pilha para o histórico de atendimentos.

- **Estruturas de Controle:** `if-else`, `switch`, `for`, `while` para a lógica de atendimento e manipulação das estruturas.
- **Noções de Complexidade:** Embora não seja o foco principal, a escolha e manipulação das estruturas de dados devem considerar a eficiência básica das operações.

Funcionalidades Mínimas Requeridas

O simulador deve ser capaz de (minimo):

1. **Adicionar Clientes:** Permitir a entrada de novos clientes no sistema, direcionando-os para a fila correta (comum, prioritária, gerencial).
2. **Chamar Próximo Cliente:** Implementar a lógica para que um atendente chame o próximo cliente disponível, respeitando as prioridades e a ordem de chegada (FIFO).
3. **Simular Atendimento:** Remover o primeiro cliente da fila (respeitando a prioridade) para simular o atendimento.
4. **Gerar Relatório Básico:** Ao final da simulação ou sob demanda, apresentar:
    - Número total de clientes atendidos.
    - Número de clientes ainda em cada fila.

Critérios de Avaliação

O trabalho será avaliado com base nos seguintes critérios:

1. **Implementação Correta das Estruturas de Dados (50%):**
    - Uso adequado e correto de Filas (Queue) e Pilhas (Stack).
    - Implementação eficiente da lógica de fila de prioridade.
2. **Funcionalidade e Usabilidade (25%):**
    - Todas as funcionalidades mínimas requeridas implementadas e funcionando corretamente.
    - Interface via console (utilizando `Scanner`) clara e intuitiva para interação com o simulador.
    - Robustez do sistema (tratamento de casos de erro básicos, como filas vazias).
3. **Qualidade do Código (25%):**
    - Código limpo, legível e bem formatado.
    - Comentários explicativos onde necessário (classes, métodos complexos, lógicas importantes).
    - Nomenclatura de variáveis, métodos e classes significativa.

Entrega e Apresentação

- **Entrega:** O código-fonte completo do projeto Java em um arquivo compactado (`.zip`).
- **Documentação:** Um arquivo `README.md` detalhado explicando o funcionamento do simulador, como compilá-lo e executá-lo, e as escolhas de design e implementação.

Prazo Sugerido

O prazo máximo para a conclusão deste trabalho esta no moodle.