1. Sistema de Pedidos Online

Descrição: Implemente um sistema de pedidos online que simula múltiplos usuários fazendo pedidos simultaneamente. Utilize threads para simular os usuários e sincronização para garantir a integridade dos dados.

Requisitos:

- Crie uma classe Order que represente um pedido com atributos como ID, item, quantidade e data.
- Crie uma classe OrderSystem que mantenha uma lista de pedidos e métodos para adicionar e processar pedidos.
- Implemente a classe UserThread que estende Thread e simule a criação de pedidos por diferentes usuários.
- Utilize sincronização para garantir que a adição de pedidos ao sistema seja thread-safe.
- Implemente um método para exibir todos os pedidos processados.

Instruções:

- A classe OrderSystem deve ter um campo List<Order> para armazenar os pedidos e métodos sincronizados para adicionar e processar pedidos.
- A classe UserThread deve simular a criação de pedidos e adicionar ao OrderSystem.
- Crie um método main que inicialize o sistema de pedidos e inicie múltiplas UserThread para simular pedidos concorrentes.

2. Sistema de Reservas de Hotel

Descrição: Desenvolva um sistema de reservas de hotel que permita múltiplos clientes reservarem quartos simultaneamente. Utilize threads para simular os clientes e sincronização para garantir que um quarto não seja reservado por mais de um cliente ao mesmo tempo.

Requisitos:

- Crie uma classe Room que represente um quarto com atributos como número do quarto e status de disponibilidade.

- Crie uma classe Hotel que contenha uma lista de quartos e métodos para reservar quartos.
- Implemente a classe ClientThread que estende Thread e simule a reserva de quartos por diferentes clientes.
- Utilize sincronização para garantir que a reserva de quartos seja thread-safe.
- Implemente um método para exibir o status de todos os quartos.

Instruções:

- A classe Hotel deve ter um campo List<Room> para armazenar os quartos e métodos sincronizados para reservar quartos.
- A classe ClientThread deve simular a reserva de quartos e interagir com o Hotel.
- Crie um método main que inicialize o hotel e inicie múltiplas ClientThread para simular reservas concorrentes.

3. Sistema de Votação Online

Descrição: Implemente um sistema de votação online que permita múltiplos usuários votarem simultaneamente. Utilize threads para simular os usuários e sincronização para garantir a integridade dos dados de votação.

Requisitos:

- Crie uma classe Vote que represente um voto com atributos como ID do usuário, opção escolhida e data.
- Crie uma classe VotingSystem que mantenha uma lista de votos e métodos para registrar votos.
- Implemente a classe UserThread que estende Thread e simule a votação por diferentes usuários.
- Utilize sincronização para garantir que a adição de votos ao sistema seja thread-safe.
- Implemente um método para exibir os resultados da votação.

Instruções:

- A classe VotingSystem deve ter um campo List<Vote> para armazenar os votos e métodos sincronizados para registrar votos.

- A classe UserThread deve simular a votação e adicionar votos ao VotingSystem.
- Crie um método main que inicialize o sistema de votação e inicie múltiplas UserThread para simular votações concorrentes.

4. Sistema de Transferência de Arquivos

Descrição: Desenvolva um sistema de transferência de arquivos que permita múltiplos usuários transferirem arquivos simultaneamente. Utilize threads para simular os usuários e sincronização para garantir a integridade dos dados durante a transferência.

Requisitos:

- Crie uma classe FileTransfer que represente uma transferência de arquivo com atributos como nome do arquivo, tamanho e status.
- Crie uma classe FileServer que contenha métodos para iniciar e monitorar transferências de arquivos.
- Implemente a classe UserThread que estende Thread e simule a transferência de arquivos por diferentes usuários.
- Utilize sincronização para garantir que a transferência de arquivos seja thread-safe.
- Implemente um método para exibir o status de todas as transferências.

Instruções:

- A classe FileServer deve ter métodos sincronizados para iniciar e monitorar transferências de arquivos.
- A classe UserThread deve simular a transferência de arquivos e interagir com o FileServer.
- Crie um método main que inicialize o servidor de arquivos e inicie múltiplas UserThread para simular transferências concorrentes.

5. Processamento Paralelo de Dados de Sensores

Descrição: Implemente um sistema que processe dados de sensores em tempo real utilizando programação paralela. Utilize threads para processar os dados de diferentes sensores simultaneamente.

Requisitos:

- Crie uma classe SensorData que represente os dados de um sensor com atributos como ID do sensor, valor medido e data.
- Crie uma classe SensorProcessor que implemente a interface Callable<Double> e processe os dados de um sensor para calcular a média dos valores medidos.
- Crie uma classe ParallelSensorProcessor que gerencie a execução paralela do processamento dos dados de múltiplos sensores.
- Utilize um ExecutorService com um pool de threads para processar os dados em paralelo.
- Agregue os resultados de todas as threads para obter a média geral dos valores medidos.

Instruções:

- A classe SensorProcessor deve ter um construtor que aceite uma lista de dados de sensores.
- Implemente o método call da interface Callable<Double> para calcular a média dos valores medidos.
- A classe ParallelSensorProcessor deve inicializar um ExecutorService com um pool de threads fixo e submeter múltiplas instâncias de SensorProcessor para processamento.
- Crie um método main que inicialize a ParallelSensorProcessor, passe uma lista de dados de sensores e exiba a média geral dos valores medidos após o processamento.

DESAFIOS

1. Simulação de Rede Social com Processamento de Feed

Descrição: Desenvolva uma simulação de uma rede social que permita múltiplos usuários postarem, curtirem e comentarem simultaneamente. Além disso, implemente um sistema de processamento de feed que atualize o feed dos usuários em tempo real utilizando programação paralela.

Requisitos:

- Crie classes Post, Comment e Like para representar postagens, comentários e curtidas, respectivamente.

- Crie uma classe User que represente um usuário com atributos como ID, nome, lista de amigos, posts, comentários e curtidas.
- Crie uma classe SocialNetwork que contenha um conjunto de usuários e métodos para adicionar posts, comentários e curtidas.
- Implemente uma classe UserThread que estenda Thread e simule as atividades dos usuários na rede social.
- Utilize um ExecutorService para gerenciar threads que atualizam o feed dos usuários em tempo real.
- Implemente um mecanismo para garantir a consistência dos dados durante as operações concorrentes, utilizando sincronização adequada.
- Garanta que o feed dos usuários seja atualizado corretamente após cada interação (post, comentário ou curtida).

Instruções:

- A classe SocialNetwork deve ter um campo Map<Integer, User> para armazenar os usuários e métodos sincronizados para adicionar posts, comentários e curtidas.
- A classe UserThread deve simular as atividades dos usuários, incluindo postagens, curtidas e comentários, interagindo com a SocialNetwork.
- Implemente uma classe FeedUpdater que implemente a interface Callable<Void> e atualize o feed dos usuários em paralelo.
- Crie um método main que inicialize a rede social, adicione usuários e inicie múltiplas UserThread para simular interações concorrentes.

2. Processamento Paralelo de Big Data com Hadoop e MapReduce

Descrição: Implemente um sistema de processamento de big data utilizando o framework Hadoop e o modelo de programação MapReduce. A tarefa é processar grandes volumes de dados de transações financeiras e calcular estatísticas como total de transações, média de valores e detectar possíveis fraudes.

Requisitos:

- Configure um ambiente Hadoop e implemente um job MapReduce para processar dados de transações.
- Crie uma classe Transaction para representar uma transação financeira com atributos como ID, valor, data, e ID do usuário.

- Implemente o Mapper para ler e mapear os dados de transações, emitindo chaves e valores apropriados.
- Implemente o Reducer para agregar os dados mapeados e calcular as estatísticas desejadas.
- Utilize programação paralela para dividir a carga de processamento entre múltiplos nós do cluster Hadoop.
- Implemente um mecanismo para detectar possíveis fraudes, como transações acima de um limite pré-definido, utilizando programação paralela.

Instruções:

- A classe TransactionMapper deve estender Mapper<LongWritable, Text, Text, DoubleWritable> e implementar o método map para processar cada linha do arquivo de entrada.
- A classe TransactionReducer deve estender Reducer<Text, DoubleWritable, Text, DoubleWritable> e implementar o método reduce para agregar os valores e calcular as estatísticas.
- Configure o job MapReduce no Hadoop e execute-o em um cluster de múltiplos nós.
- Crie um método main que configure e inicie o job MapReduce, fornecendo o caminho dos arquivos de entrada e saída, e exiba as estatísticas calculadas após o processamento.