Universidade Federal do Amazonas

Departamento de Computação

Faculdade de Tecnologia

Trabalho Prático 02 – Barbearia do Recruta Zero

Disciplina: IEC584 – Sistemas Operacionais

Curso: FT05 – Engenharia da Computação

Docente: Eduardo Luzeiro Feitosa

Discente: Laura Aguiar Martinho **Matrícula:** 21952064 **Turma:** EC01
Lorena Bastos Amazonas **Matrícula:** 21952638 **Turma:** EC01

Introdução

A barbearia do Recruta Zero é um sistema implementado em um forte americano nos arredores de San Francisco, frequentado por oficiais, sargentos e cabos. Este documento descreve a implementação do sistema, que resolve um problema semelhante ao "problema do barbeiro dorminhoco". O sistema utiliza threads e semáforos (pthreads) para funcionar e inclui várias regras e funcionalidades para fazer com que o barbeiro atenda todos os seus clientes.

Regras e Funcionalidades

Sargento Tainha: É um sargento gordo e dorminhoco que, periodicamente, entre um cochilo e outro, tenta adicionar um (e somente um) novo cliente à fila de cadeiras da barbearia (buffer) caso haja lugar disponível. A periodicidade do cochilo é variável e determinada pelo usuário através da linha de comando (entre 1 e 5 segundos).

A Barbearia: Os clientes compartilham um conjunto de 20 cadeiras com distribuição das categorias de oficiais, sargentos e cabos em três filas FIFO, sendo uma fila para cada categoria. Somente um cliente pode ser atendido por vez por cada um dos barbeiros (quando houver mais de um).

Recruta Zero (Barbeiro): É o responsável pela barbearia. Cada corte de cabelo pode durar entre 4 e 6 segundos no caso de um oficial, de 2 a 4 segundos no caso de um sargento e de 1 a 3 segundos no caso de um cabo. A prioridade de atendimento segue a ordem: oficiais, sargentos e cabos.

Tenente Escovinha: A pedido do General Dureza, o Tenente Escovinha é responsável por fornecer um relatório das atividades da barbearia, a cada 3 segundos o tenente verifica o estado da barbearia e ao final de cada dia elabora um relatório que inclui informações sobre a ocupação das cadeiras, o comprimento médio das filas, o tempo médio de atendimento e de espera por categoria, o número de atendimentos e o número de clientes por categoria (oficiais, sargentos, cabos e pausa).

Entrada de Dados

A entrada de dados é gerada aleatoriamente em um formato que inclui a categoria do cliente (oficial, sargento, cabo ou pausa) e o tempo de serviço. Uma pausa (categoria 0) indica que não há ninguém na fila.

Término da Execução

O programa encerra quando não há mais clientes esperando por três tentativas consecutivas do Sargento Tainha. Nesse ponto, o Sargento pode ir imediatamente para casa, mas os barbeiros devem concluir o atendimento dos clientes restantes.

Resultados

O sistema é testado em três casos diferentes para a geração do relatório pelo Tenente Escovinha:

Caso A: O Recruta Zero atende todas as filas de clientes, seguindo a ordem de prioridade (oficiais, sargentos e cabos).

Caso B: O Recruta Zero é auxiliado por outro barbeiro (Dentinho), e ambos atendem todas as filas, seguindo a ordem de prioridade (oficiais, sargentos e cabos).

Caso C: Três barbeiros (Recruta Zero, Dentinho e Otto) atendem exclusivamente a uma fila cada, mas podem ajudar outras filas conforme a disponibilidade e ainda seguindo a ordem de prioridade (oficiais, sargentos e cabos).

Código

Classe Barber

A classe Barber representa um barbeiro na barbearia, responsável por atender os clientes e manter métricas detalhadas de desempenho.

Atributos

private BarbersName barberID: Este atributo armazena o identificador do barbeiro, que é do tipo *BarbersName*.

Barbershop barbershop: Uma referência à barbearia em que o barbeiro está trabalhando.

public static Metrics totalServiceTime: Armazena métricas de tempo total de serviço por categoria.

public static Metrics totalWaitingTime: Armazena métricas de tempo total de espera por categoria.

public static Metrics servicesAmount: Armazena métricas de quantidade de serviços realizados por categoria.

Construtor

Barber(BarbersName barberID, Barbershop barbershop): Este é o construtor da classe *Barber*, que recebe o identificador do barbeiro e a referência à barbearia como parâmetros. Ele inicializa as métricas com objetos *Metrics* vazios.

Método ‘run()’

Este método é implementado a partir da interface *Runnable* e é executado quando uma instância de *Barber* é executada em um thread separado.

Inicialmente, o método exibe uma mensagem informando que o barbeiro chegou à barbearia.

Dentro de um loop infinito, o código verifica se há clientes nas filas e se o "Sargento Tainha" terminou sua execução.

Se não houver clientes nas filas, o barbeiro aguarda a chegada de clientes usando um semáforo.

O código seleciona o próximo cliente a ser atendido com base nas regras específicas de cada cenário, como a fila de clientes ou a categoria do barbeiro.

Depois de selecionar um cliente, o código calcula o tempo de espera do cliente, o tempo de serviço e atualiza as métricas correspondentes.

Em seguida, o código simula o corte de cabelo, aguardando o tempo necessário.

Após o corte de cabelo, o cliente é liberado usando o semáforo *Barbershop.customersSemaphore*.

O loop continua até que o barbeiro saia da barbearia quando não há clientes nas filas e o "Sargento Tainha" tenha terminado.

Classe Barbershop

A classe Barbershop representa a barbearia e seu funcionamento, incluindo a gestão de filas de clientes, controle de acesso a recursos compartilhados e execução de threads de clientes e barbeiros.

Atributos

public static final double CHAIRS_AMOUNT = 20.0: Este atributo define o número total de cadeiras disponíveis na barbearia.

private List<Thread> customerThreadList: Uma lista que mantém as threads dos clientes na barbearia.

private List<Thread> barberThreadList: Uma lista que mantém as threads dos barbeiros na barbearia.

public static int tainhaSleepingTime: Armazena o tempo de sono do "Sargento Tainha".

public static Semaphore customersSemaphore: Um semáforo que controla o acesso dos clientes à barbearia.

public static Semaphore barbersSemaphore: Um semáforo que controla o acesso dos barbeiros à barbearia.

public static Semaphore mutex: Um semáforo que é usado para controlar o acesso às filas de clientes.

public static int barbAmount: Armazena o número de barbeiros na barbearia.

private List<Customer> customersList: Uma lista que mantém os clientes que serão atendidos na barbearia.

public Queue<Customer> officerQueue: Uma fila para clientes da categoria "Oficial".

public Queue<Customer> sergeantQueue: Uma fila para clientes da categoria "Sargento".

public Queue<Customer> corporalQueue: Uma fila para clientes da categoria "Cabo".

private Map<BarbersName, Queue<Customer>> barberQueues: Um mapeamento que associa cada categoria de barbeiro a sua respectiva fila de clientes.

public static boolean barberShopIsClosed: Indica se a barbearia está fechada.

public int nOfficer: Contador de clientes da categoria "Oficial".

public int nSergeant: Contador de clientes da categoria "Sargento".

public int nCorporal: Contador de clientes da categoria "Cabo".

public int nBreak: Contador de clientes em pausa.

public char barbershopCase: Armazena o caso de teste que define o comportamento da barbearia.

Construtor

public Barbershop(): O construtor da classe *Barbershop* inicializa várias estruturas de dados, incluindo as listas de threads de clientes e barbeiros, semáforos para controle de acesso e filas de clientes para cada categoria.

Métodos

public void setBarbershopCase(char barbershopCase): Este método permite definir o caso de teste para a barbearia.

public void setCustomersList(List customersList): Define a lista de clientes que serão atendidos na barbearia.

public char getBarbershopCase(): Retorna o caso de teste atual da barbearia.

public List getCustomersList(): Retorna a lista de clientes que serão atendidos.

public List getBarberThreadList(): Retorna a lista de threads dos barbeiros.

public List getCustomerThreadList(): Retorna a lista de threads dos clientes.

public Customer getNextCustomer(BarbersName barberID): Obtém o próximo cliente a ser atendido com base no nome do barbeiro.

public void start(): Inicia a operação da barbearia com base no caso de teste fornecido e cria threads para os barbeiros e os processos "Escovinha" e "Sargento Tainha".

Classe Customer

A classe Customer representa um cliente da barbearia e contém informações sobre a categoria do cliente, tempo necessário para o corte de cabelo, identificador único, horários de chegada e saída, tempo total gasto na barbearia e uma referência à instância da barbearia onde o cliente será atendido.

Atributos

private CustomerCategory cutomerCategory: Este atributo armazena a categoria do cliente, que pode ser "Oficial," "Sargento," ou "Cabo."

private int cutHairTime: Representa o tempo necessário para cortar o cabelo do cliente, em segundos.

private int id: Identificador único atribuído a cada cliente.

private Instant startTime: Armazena a hora de chegada do cliente à barbearia.

private Instant endTime: Armazena a hora de saída do cliente da barbearia.

private Duration elapsedTime: Armazena o tempo total gasto pelo cliente na barbearia, calculado como a diferença entre startTime e endTime.

private Barbershop barbershop: Uma referência à instância da barbearia onde o cliente será atendido.

Construtor

public Customer(Barbershop barbershop): O construtor da classe Customer recebe uma instância da classe Barbershop como parâmetro, permitindo que o cliente seja associado a uma barbearia específica.

Métodos

public CustomerCategory getCutomerCategory(): Retorna a categoria do cliente.

public void setCutomerCategory(CustomerCategory category): Define a categoria do cliente com base em um valor da enumeração CustomerCategory.

public void setCategory(int category): Define a categoria do cliente com base em um valor inteiro, onde 0 representa "Oficial," 1 representa "Sargento," e 2 representa "Cabo."

public int getCutHairTime(): Retorna o tempo necessário para cortar o cabelo do cliente.

public void setCutHairTime(int serviceTime): Define o tempo necessário para cortar o cabelo do cliente.

public int getId(): Retorna o identificador único do cliente.

public void setId(int id): Define o identificador único do cliente.

public Instant getStartTime(): Retorna a hora de chegada do cliente à barbearia.

public void setStartTime(Instant startTime): Define a hora de chegada do cliente à barbearia.

public Instant getEndTime(): Retorna a hora de saída do cliente da barbearia.

public void setEndTime(Instant endTime): Define a hora de saída do cliente da barbearia.

public Duration getElapsedTime(): Retorna o tempo total gasto pelo cliente na barbearia.

public void setElapsedTime(Duration elapsedTime): Define o tempo total gasto pelo cliente na barbearia.

@Override public void run(): Implementa o método run() da interface *Runnable*. Este método é executado quando a thread do cliente é iniciada e representa o comportamento do cliente na barbearia.

Exibe uma mensagem informando que o cliente chegou à barbearia e exibe seu identificador.

Registra o horário de chegada do cliente.

Obtém a categoria do cliente.

Adiciona o cliente à fila apropriada na barbearia com base em sua categoria.

Libera um barbeiro para atender o cliente.

Aguarda a conclusão do atendimento na barbearia.

Exibe uma mensagem informando que o cliente saiu da barbearia e exibe seu identificador.

Classe Escovinha

A classe Escovinha é responsável por calcular e exibir métricas relacionadas à ocupação da barbearia, comprimento das filas e tempo médio de atendimento e espera por categoria de cliente.

Atributos

private Metrics totalOcupationState: Armazena as métricas de ocupação total da barbearia.

private Metrics totalQueueSizes: Armazena as métricas de comprimento total das filas.

private int readsAmount: Conta a quantidade de leituras realizadas pelo objeto Escovinha.

Barbershop barbershop: Referência à instância da barbearia associada ao objeto Escovinha.

Construtor

public Escovinha(Barbershop barbershop): Construtor que recebe uma instância da classe Barbershop como parâmetro para associar o objeto Escovinha a uma barbearia específica.

Método ‘run()’

O método run() é implementado a partir da interface *Runnable* e representa o comportamento do objeto Escovinha em sua thread.

Realiza leituras periódicas das métricas relacionadas à ocupação e filas da barbearia.

Calcula as métricas de ocupação atual da barbearia, convertendo-as em porcentagens com base no número de cadeiras disponíveis.

Exibe o estado de ocupação atual da barbearia, mostrando a porcentagem de ocupação por categoria de cliente.

Atualiza as métricas totais com as métricas atuais.

Aguarda um intervalo de 3 segundos antes de realizar a próxima leitura.

Quando a barbearia é fechada (indicado por Barbershop.barberShopIsClosed), o objeto Escovinha calcula médias das métricas totais e exibe métricas finais.

Exibe a quantidade total de leituras realizadas pelo objeto Escovinha.

Exibe o estado de ocupação médio das cadeiras em porcentagem, calculado a partir das métricas totais.

Exibe o comprimento médio das filas para cada categoria de cliente.

Exibe o tempo médio de atendimento por categoria de cliente.

Exibe o tempo médio de espera por categoria de cliente.

Exibe o número total de clientes por categoria.

Classe Tainha

A classe Tainha é responsável por controlar o atendimento dos clientes na barbearia e adicionar clientes na fila de atendimento quando apropriado.

Atributos

private static boolean finished: Indica se a execução do Sargento Tainha foi concluída.

Barbershop barbershop: Referência à instância da barbearia associada ao objeto Tainha.

Construtor

public Tainha(Barbershop barbershop): Construtor que recebe uma instância da classe Barbershop como parâmetro para associar o objeto Tainha a uma barbearia específica.

public static boolean isFinished(): Método estático que retorna o valor do atributo finished, indicando se a execução do Sargento Tainha foi concluída.

Método 'run()'

O método run() é implementado a partir da interface *Runnable* e representa o comportamento do objeto Tainha em sua thread.

Exibe uma mensagem informando que o Sargento Tainha chegou à barbearia.

Inicializa variáveis para controlar a categoria do cliente atual e a quantidade de clientes da categoria "PAUSE" consecutivos.

Entra em um loop principal que controla o comportamento do Sargento Tainha enquanto ele está em execução.

Dentro do loop, o Sargento Tainha "dorme" por um período de tempo determinado antes de realizar qualquer ação.

Verifica a categoria do cliente atual na fila da barbearia.

Conta o número de clientes consecutivos que têm categoria igual a "PAUSE".

Se o cliente não estiver em pausa, inicia um novo thread para atendê-lo na barbearia e exibe uma mensagem informando que o Sargento Tainha adicionou um novo cliente.

Reinicia o contador de clientes em pausa, já que um cliente não em pausa foi atendido.

Move para o próximo cliente na fila.

Verifica se o número total de clientes na fila da barbearia atingiu o limite das cadeiras disponíveis e, se sim, passa para o próximo cliente sem iniciar um novo thread para ele.

Após o loop, sinaliza que o Sargento Tainha terminou sua execução, alterando o valor do atributo finished para true, e exibe uma mensagem informando que o Sargento Tainha saiu da barbearia.

Classe Metrics

A classe Metrics é usada para manter métricas relacionadas à ocupação da barbearia, incluindo a quantidade de clientes em diferentes categorias e o número de cadeiras vazias.

Atributos

private double officer: Representa a métrica relacionada à quantidade de clientes da categoria "Oficial".

private double sergeant: Representa a métrica relacionada à quantidade de clientes da categoria "Sargento".

private double corporal: Representa a métrica relacionada à quantidade de clientes da categoria "Cabo".

private double empty: Representa a métrica relacionada à quantidade de cadeiras vazias na barbearia.

Construtores

public Metrics(double officer, double sergeant, double corporal): Construtor que recebe a quantidade de clientes de cada categoria e calcula automaticamente a quantidade de cadeiras vazias.

public Metrics(): Construtor padrão que inicializa todas as métricas como zero.

Métodos

public void add(Metrics other): Adiciona as métricas de outra instância da classe Metrics à instância atual, atualizando os valores das métricas.

public void divide(double num): Divide todas as métricas pela quantidade especificada (num) e atualiza os valores das métricas.

public Metrics divideBy(double num): Retorna uma nova instância da classe Metrics na qual todas as métricas foram divididas pela quantidade especificada (num).

Classe Main

A classe Main é a classe principal do programa e contém o método main, que é o ponto de entrada do programa. Esta classe é responsável por configurar a barbearia, ler os argumentos da linha de comando e criar clientes com base na entrada fornecida. odo start da instância barbershop. Fecha o scanner após a leitura de todos os dados.

Método Main

O método main é o ponto de entrada do programa e é executado quando o programa é iniciado.

Dentro do método main, o programa realiza as seguintes ações:

Cria uma instância da classe *Barbershop* chamada *barbershop* para representar a barbearia.

Cria uma lista de clientes chamada customers para armazenar os clientes que serão criados posteriormente.

Define o caso de teste padrão da barbearia como 'A'.

Lê os argumentos da linha de comando, onde o primeiro argumento é o tempo de sono do "Sargento Tainha" e o segundo argumento é o caso de teste da barbearia. Se não houver dois argumentos fornecidos, o programa exibe uma mensagem de erro e sai.

Enquanto houver entrada disponível no scanner, o programa lê pares de números inteiros, onde o primeiro número representa a categoria do cliente e o segundo número representa o tempo necessário para cortar o cabelo desse cliente.

Para cada par de números lidos, o programa cria uma instância da classe *Customer* com base na categoria e no tempo de corte lidos, atribui um ID único a cada cliente e adiciona o cliente à lista de clientes customers.

Define o caso de teste da barbearia como o valor lido a partir dos argumentos da linha de comando.

Define a lista de clientes da barbearia como a lista de clientes criada anteriormente.

Inicia a execução da barbearia chamando o método start da instância *barbershop*.

Fecha o scanner após a leitura de todos os dados.

Execução do Projeto

Para executar o projeto, utilizar o seguinte comando:

```
javac -d ./bin -sourcepath ./src src/Main.java && java -cp bin Main < src/test/Data1
```

O arquivo Data1 tem a seguinte estrutura:

Primeira Linha: <tempo de sono do Sargento Tainha> <caso de teste>

Segunda Linha: <categoria do cliente> <tempo do corte de cabelo>

Caso de Teste: A, B ou C

Categoria do Cliente: 0 – pausa, 1 – oficial, 2 – sargento, 3 – cabo

Tempo de Corte de Cabelo: 1 – 6 segundos, dependendo da categoria do cliente

Exemplo:

1 A

1 4

1 4

1 4

O tempo de sono do Sargento Tainha é de 1s e o caso é A. Há três oficiais com corte de cabelo de 4 segundos.