



# VIRTUAL QUEUE

PROJETO 2VA - PARADIGMAS DE PROGRAMAÇÃO

PROFESSOR: SIDNEY DE CARVALHO

ALUNO: ANTONY ALBUQUERQUE



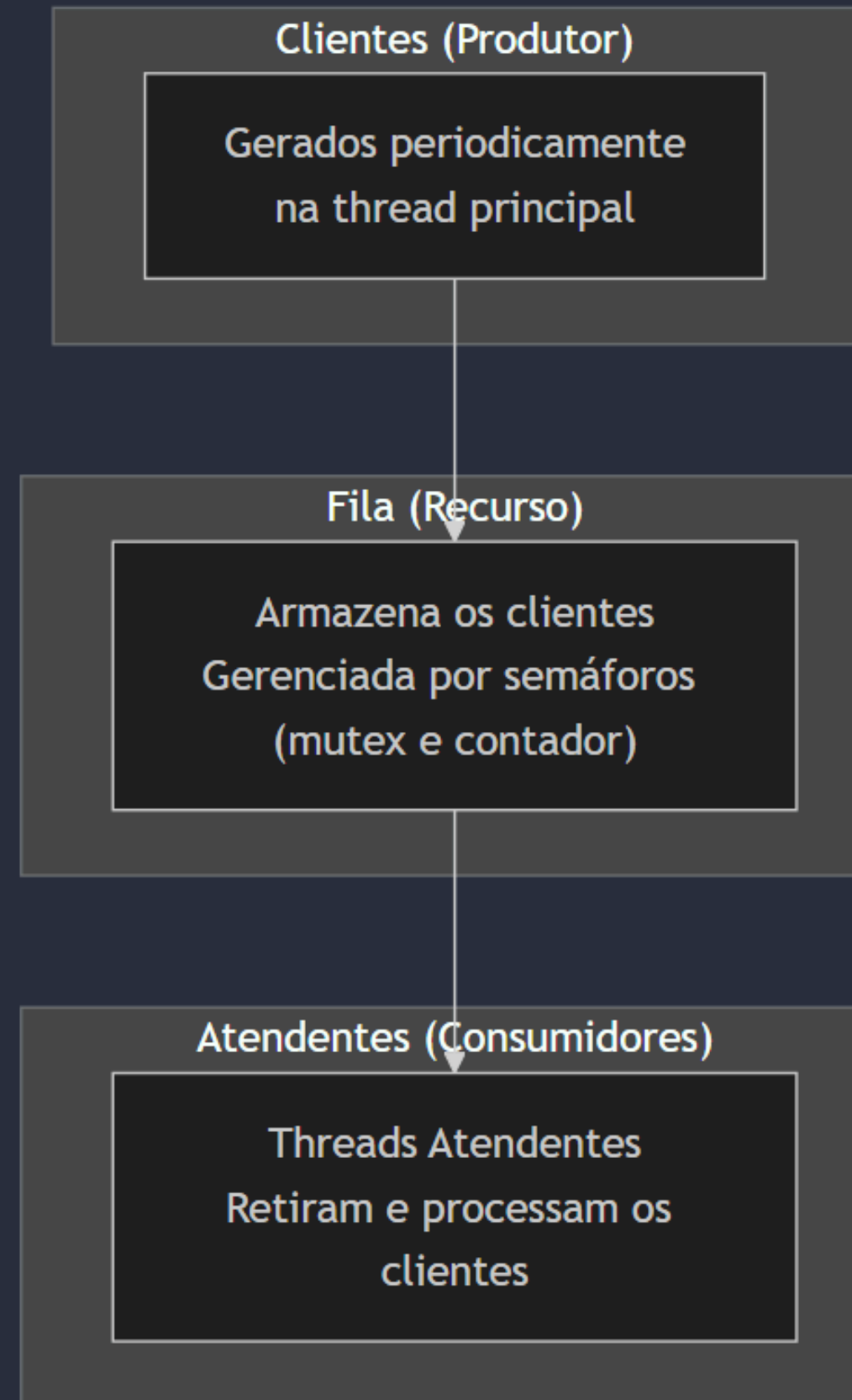
# Descrição do projeto

- Projeto "Virtual Queue" simula um sistema de atendimento no modelo produtor-consumidor, onde clientes são gerados periodicamente (produtor) e inseridos em uma fila compartilhada.
- Múltiplas threads atendedoras (consumidoras) retiram e processam esses clientes utilizando semáforos para garantir a sincronização, enquanto uma thread monitor exibe periodicamente o status do sistema via console.
- Feito em java
- Ferramenta usada IDE Eclipse



# Fluxo do funcionamento

- Clientes (produtor)
- Fila (recurso compartilhado)
- Atendentes (consumidores)












# Principais Componentes do Código

- **GerenciadorDeAtendimentos:** Gerencia o fluxo geral do sistema (main)
- **FilaSemaforo:** Controla o acesso à fila usando semáforos
- **Atendente:** Threads que consomem clientes e atualizam estatísticas
- **Monitor:** Thread que monitora e exibe informações do sistema
- **Estatisticas:** Registra e calcula dados de atendimento

## TODAS AS CLASSES

- >  `Atendente.java`
- >  `Cliente.java`
- >  `Config.java`
- >  `Estatisticas.java`
- >  `FilaSemaforo.java`
- >  `GerenciadorDeAtendimentos.java`
- >  `Monitor.java`



# Controle de Concorrência

- Foram utilizados semáforos para gerenciar o acesso à fila compartilhada.
- Um semáforo binário (mutex) garante que apenas uma thread modifique a fila por vez, evitando condições de corrida.
- Um semáforo contador (items) controla quantos itens estão disponíveis, bloqueando as threads consumidoras se a fila estiver vazia.



# FilaSemaforo

- Gerencia uma fila compartilhada com semáforos.
- O mutex (semáforo binário) protege a seção crítica (inserir/retirar).
- O semáforo contador garante que consumidores só retirem quando houver itens.

MAR 2025

```
1 import java.util.LinkedList;
2
3 import java.util.concurrent.Semaphore;
4
5 /*
6  FilaSemaforo: Classe responsável por gerenciar uma fila de clientes com controle de sincronização.
7
8  - mutex: Semáforo binário (inicializado com 1 permissão) que garante exclusão mútua,
9  permitindo que apenas uma thread acesse a região crítica (adição ou remoção de clientes) de cada vez.
10
11  - items: Semáforo de contagem (inicializado com 0) que controla quantos itens estão disponíveis na fila.
12  Isso evita que uma thread tente remover um cliente quando a fila está vazia, pois a thread ficará bloqueada
13  até que um cliente seja inserido (quando items.release() é chamado).
14 */
15
16 public class FilaSemaforo {
17
18     private final LinkedList<Cliente> fila = new LinkedList<>();
19     private final Semaphore mutex = new Semaphore(1);
20     private final Semaphore items = new Semaphore(0);
21
22     // o produtor vai inserir os clientes na fila
23     public void colocarNaFila(Cliente cliente) throws InterruptedException {
24         mutex.acquire();
25         try {
26             fila.add(cliente); // <- SEÇÃO CRÍTICA (acessando recurso compartilhado)
27         } finally {
28             mutex.release();
29         }
30         items.release();
31     }
32
33     // consumidor vai retirar da fila caso exist algum cliente na fila, caso não exista ele fica bloqueado
34     public Cliente retirarDaFila() throws InterruptedException {
35         items.acquire();
36         mutex.acquire();
37         try {
38             return fila.removeFirst(); // <- SEÇÃO CRÍTICA (removendo recurso compartilhado)
39         } finally {
40             mutex.release();
41         }
42     }
43
44     public int filaEspera() {
45         int count = 0;
46         try {
47             mutex.acquire();
48             for (Cliente c : fila) {
49                 if (!c.isPoison()) {
50                     count++;
51                 }
52             }
53         } catch (InterruptedException e) {
54             Thread.currentThread().interrupt();
55         } finally {
56             mutex.release();
57         }
58         return count;
59     }
60 }
61
62
63
```



# Cliente

- Representa um cliente, com id e tempo de chegada.
- Possui um construtor normal e outro "poison pill" (id = -1) para sinalizar o término.
- A poison pill é empregada para sinalizar aos consumidores que não há mais trabalho a ser processado.
  - Indicando que as threads consumidoras devem encerrar sua execução.
  - poison pill é um termo bastante usado para programação concorrente!!

```

1
2  /*
3   Cliente: Representa um cliente a ser atendido.
4   - id identifica o cliente; id = -1 indica a "poison pill" (sinal de término).
5   - tempoChegada registra o instante de criação do cliente.
6  */
7
8  public class Cliente {
9      private int id;
10     private long tempoChegada;
11
12     // Construtor normal para clientes
13     public Cliente(int id) {
14         this.id = id;
15         this.tempoChegada = System.currentTimeMillis();
16     }
17
18     // Construtor para "poison pill": um cliente com id = -1 sinaliza término.
19     public Cliente() {
20         this.id = -1;
21         this.tempoChegada = 0;
22     }
23
24     public int getId() {
25         return id;
26     }
27
28     public long getTempoChegada() {
29         return tempoChegada;
30     }
31
32     // Retorna true se este cliente for a "poison pill"
33     public boolean isPoison() {
34         return id == -1;
35     }
36
37     // Retorna "CondicaoParada" se o cliente for uma poison pill; caso contrário, retorna "Cliente " seguido do id.
38     @Override
39     public String toString() {
40         if (isPoison()) {
41             return "CondicaoParada";
42         }
43         return "Cliente " + id;
44     }
45 }
46

```



# Atendente

- Gerencia uma fila de clientes usando semáforos para sincronizar o acesso.
- Um semáforo (mutex) garante exclusão mútua, enquanto outro (itens) controla quantos clientes estão disponíveis.

```

1  /*
2   Atendente: Thread que processa clientes da fila.
3
4   No run():
5   - Loop infinito que retira clientes (bloqueia se vazia).
6   - Se o cliente é "poison pill", reinsere-o e encerra.
7   - Calcula o tempo de espera, simula atendimento com sleep e atualiza estatísticas.
8   - Trata interrupções.
9  */
10
11
12 public class Atendente extends Thread {
13
14     private int id;
15     private FilaSemaforo fila;
16     private Estatisticas estatisticas;
17
18     public Atendente(int id, FilaSemaforo fila, Estatisticas estatisticas) {
19         this.id = id;
20         this.fila = fila;
21         this.estatisticas = estatisticas;
22     }
23
24     @Override
25     public void run() {
26         try {
27             while (true) {
28
29                 // Retira o cliente da fila; se a fila estiver vazia, retirar() bloqueia até que um cliente seja inserido.
30                 Cliente cliente = fila.retirarDaFila();
31                 System.out.println();
32
33                 // Verifica se recebeu a "poison pill" para finalizar o atendimento.
34                 if (cliente.isPoison()) {
35                     System.out.println("Atendente " + id + " recebeu sinal de término.");
36                     System.out.println();
37
38                     // Reinsere a poison pill para que outros atendentes também possam terminar e encerra seu próprio laço.
39                     fila.colocarNaFila(cliente);
40                     break;
41                 }
42
43                 long tempoInicioAtendimento = System.currentTimeMillis();
44                 long tempoEspera = tempoInicioAtendimento - cliente.getTempoChegada();
45                 System.out.println("Atendente " + id + " atendendo " + cliente + " (esperou " + tempoEspera + " ms).");
46                 System.out.println();
47
48                 // Simula o tempo de atendimento.
49                 Thread.sleep(Config.TEMPO_ATENDIMENTO);
50
51                 // Atualiza as estatísticas.
52                 estatisticas.clienteAtendido(tempoEspera);
53                 System.out.println();
54                 System.out.println("Atendente " + id + " finalizou atendimento de " + cliente + ".");
55                 System.out.println();
56             }
57         } catch (InterruptedException e) {
58             System.out.println("Atendente " + id + " interrompido.");
59         }
60         System.out.println("Atendente " + id + " finalizado.");
61     }
62 }
63

```





# Monitor

- Exibe periodicamente o estado do sistema.
- Consulta a fila e as estatísticas para informar quantos clientes estão esperando e os dados acumulados.

```

1
2  /*
3   Monitor: Thread que monitora periodicamente o estado do sistema.
4   - fila: Fila compartilhada para consultar o número de clientes aguardando.
5   - estatísticas: Dados de atendimento que são exibidos junto à quantidade de clientes na fila.
6  */
7
8  public class Monitor extends Thread {
9      private FilaSemaforo fila;
10     private Estatisticas estatisticas;
11
12     public Monitor(FilaSemaforo fila, Estatisticas estatisticas) {
13         this.fila = fila;
14         this.estatisticas = estatisticas;
15     }
16
17     @Override
18     public void run() {
19         try {
20             while (!Thread.currentThread().isInterrupted()) {
21
22                 // simular o tempo de atendimento
23                 Thread.sleep(Config.TEMPO_MONITORAMENTO);
24
25                 // Chama o método filaEspera() da FilaSemaforo para obter o número de clientes esperando na fila na fila
26                 int clientesNaFila = fila.filaEspera();
27
28                 System.out.println();
29                 System.out.println("Clientes na fila: " + clientesNaFila);
30                 System.out.println();
31                 System.out.println(estatisticas);
32                 System.out.println();
33             }
34         } catch (InterruptedException e) {
35             System.out.println("Monitor interrompido.");
36         }
37         System.out.println("Monitor finalizado.");
38     }
39 }
40
41

```



# Estatísticas

- Registra o total de clientes atendidos e o tempo de espera acumulado.
- Calcula o tempo médio de espera de forma sincronizada para evitar inconsistências.

```

1
2  /*
3   Monitor: Thread que monitora periodicamente o estado do sistema.
4   - fila: Fila compartilhada para consultar o número de clientes aguardando.
5   - estatísticas: Dados de atendimento que são exibidos junto à quantidade de clientes na fila.
6  */
7
8  public class Monitor extends Thread {
9      private FilaSemaforo fila;
10     private Estatisticas estatisticas;
11
12     public Monitor(FilaSemaforo fila, Estatisticas estatisticas) {
13         this.fila = fila;
14         this.estatisticas = estatisticas;
15     }
16
17     @Override
18     public void run() {
19         try {
20             while (!Thread.currentThread().isInterrupted()) {
21
22                 // simular o tempo de atendimento
23                 Thread.sleep(Config.TEMPO_MONITORAMENTO);
24
25                 // Chama o método filaEspera() da FilaSemaforo para obter o número de clientes esperando na fila na fila
26                 int clientesNaFila = fila.filaEspera();
27
28                 System.out.println();
29                 System.out.println("Clientes na fila: " + clientesNaFila);
30                 System.out.println();
31                 System.out.println(estatisticas);
32                 System.out.println();
33
34             }
35         } catch (InterruptedException e) {
36             System.out.println("Monitor interrompido.");
37         }
38         System.out.println("Monitor finalizado.");
39     }
40 }
41

```



# Config



```
1 public class Config {
2     // Número de clientes a serem gerados
3     public static int NUM_CLIENTES = 5;
4     // Número de atendentes que processarão os clientes
5     public static final int NUM_ATENDENTES = 2;
6     // Intervalo (em milissegundos) entre a chegada dos clientes
7     public static final int INTERVALO_CHEGADA = 0;
8     // Tempo (em milissegundos) que o atendente leva para processar um cliente
9     public static final int TEMPO_ATENDIMENTO = 4000; // 4 segundos
10    // Intervalo (em milissegundos) para a atualização do monitor
11    public static final int TEMPO_MONITORAMENTO = 3000; // 3 segundos
12 }
13
```



# GerenciadorDeAtendimentos (Main)

- Orquestra o fluxo geral do sistema.
- Cria a fila, as estatísticas, inicia as threads de atendentes e o monitor.
- Gera clientes periodicamente e insere a poison pill para sinalizar o fim dos atendimentos.

```

1  import java.util.LinkedList;
2
3  import java.util.concurrent.Semaphore;
4  /*
5      FilaSemaforo: Classe responsável por gerenciar uma fila de clientes com controle de sincronização.
6
7      - mutex: Semáforo binário (inicializado com 1 permissão) que garante exclusão mútua,
8      permitindo que apenas uma thread acesse a região crítica (adição ou remoção de clientes) de cada vez.
9
10     - items: Semáforo de contagem (inicializado com 0) que controla quantos itens estão disponíveis na fila.
11     Isso evita que uma thread tente remover um cliente quando a fila está vazia, pois a thread ficará bloqueada
12     até que um cliente seja inserido (quando items.release() é chamado).
13  */
14  */
15
16  public class FilaSemaforo {
17
18      private final LinkedList<Cliente> fila = new LinkedList<>();
19      private final Semaphore mutex = new Semaphore(1);
20      private final Semaphore items = new Semaphore(0);
21
22      // o produtor vai inserir os clientes na fila
23      public void colocarNaFila(Cliente cliente) throws InterruptedException {
24          mutex.acquire();
25          try {
26              fila.add(cliente); // <- SEÇÃO CRÍTICA (acessando recurso compartilhado)
27          } finally {
28              mutex.release();
29          }
30          items.release();
31      }
32
33      //consumidor vai retirar da fila caso exist algum cliente na fila, caso não exista ele fica bloqueado
34      public Cliente retirarDaFila() throws InterruptedException {
35          items.acquire();
36          mutex.acquire();
37          try {
38              return fila.removeFirst(); // <- SEÇÃO CRÍTICA (removendo recurso compartilhado)
39          } finally {
40              mutex.release();
41          }
42      }
43
44      public int filaEspera() {
45          int count = 0;
46          try {
47              mutex.acquire();
48              for (Cliente c : fila) {
49                  if (!c.isPoison()) {
50                      count++;
51                  }
52              }
53          } catch (InterruptedException e) {
54              Thread.currentThread().interrupt();
55          } finally {
56              mutex.release();
57          }
58          return count;
59      }
60  }

```



# Cenários Testados

- Número de clientes: 3, 5, 10 e 15
- Número de atendentes: 1, 2, 4 e 6



# OBRIGADO A TODOS !