Modelagem e Construção de Protótipo de Solução para o Problema do Salão de Beleza

HIAGO MAYK GOMES DE ARAÚJO ROCHA
LUCAS SIMONETTI MARINHO CARDOSO
RUBEM KALEBE SANTOS

Natal, Brasil
Novembro de 2015

Sumário

1 Introdução

A programação concorrente está relacionada com a atividade de construir programas de computador que incluem linhas de controle distintas, as quais podem executar simultaneamente. Dessa forma, um programa dito concorrente se diferencia de um programa dito sequencial por conter mais de um contexto de execução ativo ao mesmo tempo. As diferentes linhas de controle de um programa concorrente cooperam para a execução de uma tarefa única (finalidade do programa). Para isso, na maioria dos casos é necessário usar mecanismos que permitam a comunicação entre as linhas de controle e a sincronização das suas ações de forma a garantir a correta execução da atividade fim[1].

Além de ser muito usado para aumento de desempenho e tornar a aplicação mais dinâmica, pode-se aproveitar o paradigma da programação concorrente para modelar aspectos concorrentes do mundo real. Considere, por exemplo, um salão de beleza. Ele pode ter vários funcionários e que com a disponibilidade de recursos (tesouras, por exemplo) podem trabalhar ao mesmo tempo. Pode ser interessante simular as movimentações nesse salão afim de analisar a necessidade de contratar mais gente para exercer uma atividade, ou quem sabe, acabar com um serviço que não traz muito lucro. Portanto, a modelagem das movimentações desse empreendimento pode ser muito benéfica ao dono.

Com base nisso, neste trabalho será proposto um simulador para um salão de beleza hipotético. Este documento contém a descrição do problema abordado, a descrição da solução usada, os resultados computacionais alcançados e a implementação do simulador. Tal simulador foi implementado em linguagem Java, que além de possuir vários recursos para lidar com concorrência e interfaces gráficas, facilita o uso da aplicação desenvolvida em diferentes plataformas computacionais.

2 Descrição do problema

O problema consiste na criação e simulação de um salão de beleza, o qual foi chamado de **Salão Beleza Pura**, composto por:

- 5 cabeleireiras;
- 3 manicures;

[1] Rossetto, Silvana. "Computação Concorrente (MAB-117) Cap. I: Introdução e histórico da programação concorrente." (2012).

- 2 depiladoras;
- 1 massagista
- 2 caixas.

Além disso, são dadas as seguintes regras de negócio:

a) A chegada de clientes deve ser simulada segundo um critério aleatório de tempo de chegada entre um e outro variando de 1 a 5 unidades de tempo;

b) Os clientes devem ser atendidos na ordem de chegada e da disponibilidade dos serviços;

c) A cabeleireira alocada para realizar um corte também lava o cabelo do cliente;

d) Cada cliente pode desejar de 1 a todos os serviços oferecidos pelo salão;

e) Um cliente não deve prender outro que esteja atrás de si e que deseja um serviço que esteja disponível;

f) Todo corte deve ser sempre precedido de uma lavagem;

g) O tempo gasto em cada serviço por cada cliente deve ser gerado aleatoriamente considerando a seguinte ordem decrescente de duração: penteado, corte, depilação, pés e mãos, massagem e lavagem;

h) O preço de cada serviço é de 50 reais para penteado, 30 reais para corte, 40 reais para corte e penteado, 0 reais para lavagem, 30 reais para pedicure, 40 reais para depilação e 20 reais para massagem;

i) Em geral 30% dos clientes desejam todos os serviços, 35% desejam 4, 20% desejam 3, 10% apenas 2 e 5% apenas 1;

j) Os serviços também são procurados segundo um percentual médio de 50% para corte, 40% para penteado, 30% para pedicure, 20% para depilação, 15% para massagem;

k) A política adotada pelo dono do estabelecimento é que cada profissional recebe 40% do total faturado por ele durante o dia de trabalho;

l) O salão tem por regra de negócio otimizar o tempo do cliente, atendendo-o da melhor forma e no menor tempo possível;

m) O sistema deve apresentar um resumo do movimento e do faturamento realizado;

n) Construir uma representação na tela do monitor da movimentação nas filas de entrada, de espera por cada profissional;

3 Descrição da solução

Nesta seção apresentamos a descrição detalhada da modelagem do sistema e da implementação para cada umas das regras de negócio especificadas anteriormente.

3.1 Modelagem

A modelagem do sistema foi feita tentando obedecer os conceitos do paradigma da Programação Orientada à Objetos (POO), já que a linguagem Java, linguagem escolhida para o desenvolvimento do projeto, segue esse paradigma. Criamos várias classes com a finalidade de dividir o problema e tornar mais simples a implementação e além disso também usamos o padrão de projeto *Singleton* que é apresentado em *Gang of Four*[2]. Abaixo estão apresentadas as classes usadas para modelar o sistema, descrevemos suas responsabilidades e o mostramos o código das partes mais importantes sua estrutura:

- A classe `Funcionario` é a classe que implementa a interface `Runnable` do Java e tem os atributos e métodos comuns a todos os funcionários do sistema.

```
1   public abstract class Funcionario implements Runnable
2   {
3           protected int id;
4           protected int qtdServicos;
5           protected double totalFaturadoLiquido;
6           protected double totalFaturadoBruto;
7           protected Cliente cliente;
8           protected FilasClientes filas;
9           protected Semaphore sFilasClientes;
10          protected Semaphore sFilasCaixas;
11          protected Semaphore semResumo;
12
13          public Funcionario();
14          public Funcionario(FilasClientes f, Semaphore semFilasClientes,
                Semaphore semFilasCaixas, Semaphore semResumo, int id);
```

[2]Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gama, Richard Helm, Ralph Johson e John Vlissides

```
15              public Funcionario(FilasClientes f, Cliente c)
16              public int getId();
17              public void setId(int id);
18              public int getQtdServicos();
19              public void setQtdServicos(int qtdServicos);
20              public double getTotalFaturadoBruto();
21              public double getTotalFaturadoLiquido();
22              public void setTotalFaturadoBruto(double totalFaturado);
23              public void setTotalFaturadoLiquido(double totalFaturado);
24              public Cliente getCliente();
25              public void setCliente(Cliente cliente);
26              public FilasClientes getFilas();
27              public void setFilas(FilasClientes filas);
28              public void incrementaQtdServicos();
29              public void incrementaTotalFaturado(double d);
30              protected void reposicionaCliente();
31      }
```

- As classes `Cabeleireira`, `Caixa`, `Depiladora`, `Manicure` e `Massagista` herdam todos os atributos da classe `Funcionario` e são responsáveis por executar os pedidos dos clientes. São essas as classes que terão suas instâncias executadas pelas *threads*.

```
1       public class Cabeleireira extends Funcionario
2       {
3               public Cabeleireira(FilasClientes f, Semaphore semFilasClientes,
                    Semaphore semFilasCaixas, Semaphore semResumo, int id);
4               public Cabeleireira(FilasClientes f, Cliente c) ;
5               public void run();
6               private int numeroDaFila(Cliente cliente);
7               public String toString();
8       }
```

```
1       public class Caixa extends Funcionario
2       {
3               public Caixa(FilasClientes f, Semaphore semFilasClientes, Semaphore
                    semFilasCaixas, Semaphore semResumo, int id);
4               public Caixa(FilasClientes f, Cliente c);
5               public void run();
6               public String toString();
7       }
```

```
1       public class Depiladora extends Funcionario
2       {
3               public Depiladora(FilasClientes f, Semaphore semFilasClientes,
                    Semaphore semFilasCaixas, Semaphore semResumo, int id);
4               public Depiladora(FilasClientes f, Cliente c);
5               public void run();
```

```
6               public String toString();
7       }
```

```
1       public class Manicure extends Funcionario
2       {
3               public Manicure(FilasClientes f, Semaphore semFilasClientes,
                    Semaphore semFilasCaixas, Semaphore semResumo, int id);
4               public Manicure(FilasClientes f, Cliente c);
5               public void run();
6               public String toString();
7       }
```

```
1       public class Massagista extends Funcionario
2       {
3               public Massagista(FilasClientes f, Semaphore semFilasClientes,
                    Semaphore semFilasCaixas, Semaphore semResumo, int id);
4               public Massagista(FilasClientes f, Cliente c);
5               public void run();
6               public String toString();
7       }
```

- A classe `GeradorClientes`, como o nome já sugere, gera os clientes e os insere no final da fila de atendimento. A geração dos clientes segue o que foi descrito na especificação do projeto, onde cada cliente é gerado obedecendo um tempo aleatório entre 1 e 5 segundos.

```
1       public class GeradorClientes implements Runnable
2       {
3               private int contClientes;
4               private int qntdClientesAtendidos;
5               private Random rand;
6               private FilasClientes filas;
7
8               public GeradorClientes();
9               public GeradorClientes(FilasClientes filas);
10              public int getContClientes();
11              public void setContClientes(int contClientes);
12              public int getQntdClientesAtendidos();
13              public void setQntdClientesAtendidos(int qntdClientesAtendidos);
14              public Random getRand();
15              public void setRand(Random rand);
16              public FilasClientes getFilas();
17              public void setFilas(FilasClientes filas);
18              public void run();
19              public Cliente criaCliente();
20              public ArrayList<Integer> geraTempoServicos(int quantidade);
21      }
```

- A classe `Cliente` representa um cliente no sistema e contém como atributos o seu identificador, uma lista de todos os serviços solicitados e uma lista que contém os serviços que ainda faltam ser atendidos.

```
1       public class Cliente
2       {
3               private int id;
4               private LinkedList<Servico> servicosSolicitados;
5               private LinkedList<Servico> servicosRestantes;
6
7               public Cliente();
8               public Cliente(int id);
9               public int getId()
10              public void setId(int id);
11              public LinkedList<Servico> getServicosSolicitados();
12              public void setServicosSolicitados(LinkedList<Servico>
                    servicosSolicitados);
13              public LinkedList<Servico> getServicosRestantes();
14              public void setServicosRestantes(LinkedList<Servico>
                    servicosRestantes);
15              public Funcionario getFuncionario();
16              public void setFuncionario(Funcionario funcionario);
17              public void incluirServico(Servico s);
18              public int quantidadeServicosSolicitados();
19              public int quantidadeServicosRestantes();
20              public Servico proximoServico();
21              public void popServico();
22              public String toString();
23              public boolean equals(Object obj);
24      }
```

- A classe `FilasClientes` contém as filas nas quais os clientes serão inseridos para serem atendidos por determinados funcionários. No total, existem 6 filas, sendo que 5 filas possuem um certa prioridade entre elas e são para o atendimento aos funcionários que prestam algum serviço que foi solicitado, e 1 para o atendimento aos caixas. Um explicação mais detalhada sobre como é feita a alocação e sobre as prioridades das filas é dada na seção de implementação.

```
1       public class FilasClientes
2       {
3               private ArrayList<ArrayList<Cliente>> filasClientes;
4               private ArrayList<ArrayList<Cliente>> filasCaixas;
5               private final int numFilasClientes = 5;
6               private final int numFilasCaixas = 1;
7               private static FilasClientes instance;
8
9               private FilasClientes();
10              public static FilasClientes getInstance();
```

```
11              public ArrayList<ArrayList<Cliente>> getFilasClientes();
12              public void setFilasClientes(ArrayList<ArrayList<Cliente>>
                    filasClientes);
13              public ArrayList<Cliente> getFilaClientes(int i);
14              public void setFilaClientes(int i, ArrayList<Cliente> filaClientes);
15              public ArrayList<ArrayList<Cliente>> getFilasCaixas();
16              public void setFilasCaixas(ArrayList<ArrayList<Cliente>> filasCaixas);
17              public ArrayList<Cliente> getFilaCaixa(int i);
18              public void setFilasCaixas(int i, ArrayList<Cliente> filasCaixas);
19              public int getNumFilasClientes();
20              public int getNumFilasCaixas();
21              public void insereEmFilaClientes(int fila, Cliente c);
22              public void removeDeFilaClientes(int fila, Cliente c);
23              public void insereEmFilaCaixas(Cliente c);
24              public void removeDeFilaCaixas(Cliente c);
25              public Cliente getProxParaPenteado();
26              public Cliente getProxParaCorte();
27              public Cliente getProxParaDepilacao();
28              public Cliente getProxParaPedicure();
29              public Cliente getProxParaMassagem();
30              public Cliente getProxParaCaixa();
31      }
```

- A classe `Servico` é a abstração de um serviço solicitado por um cliente.

```
1       public abstract class Servico
2       {
3               protected TipoServico tipo;
4               protected int tempo;
5               protected double preco;
6
7               public Servico();
8               public Servico(TipoServico tipo, int tempo, double preco);
9               public TipoServico getTipo();
10              public void setTipo(TipoServico tipo);
11              public int getTempo();
12              public void setTempo(int tempo);
13              public double getPreco();
14              public void setPreco(double preco);
15              public String toString();
16              public boolean equals(Object obj);
17      }
```

- As classes `Corte`, `Penteado`, `Depilacao`, `Massagem` e `Pedicure` herdam todos os atributos da classe abstrata `Servico` e representa um serviço especifico .

```
1       public class Corte extends Servico
2       {
3               public Corte(int tempo)
4       }
```

```
1       public class Penteado extends Servico
2       {
3               public Penteado(int tempo);
4       }
```

```
1       public class Depilacao extends Servico
2       {
3               public Depilacao(int tempo)
4       }
```

```
1       public class Massagem extends Servico
2       {
3               public Massagem(int tempo)
4       }
```

```
1       public class Pedicure extends Servico
2       {
3               public Pedicure(int tempo)
4       }
```

- A classe `TipoServico` representa uma enumeração para os tipos de serviços.

```
1       public enum TipoServico
2       {
3               PENTEADO,
4               CORTE,
5               DEPILACAO,
6               PEDICURE,
7               MASSAGEM,
8               LAVAGEM;
9       }
```

- A classe `Salao` é onde são instanciadas e startadas todas as *threads* que executam funções no sistema.

```
public class Salao
{
        private List<Funcionario> funcionarios;
        private FilasClientes filas;
        private List<Thread> threadsFuncionarios;
        private GeradorClientes geradorClientes;
        private Thread tGeradorClientes;
        private Semaphore sFilasClientes;
        private Semaphore sFilasCaixas;
        private Semaphore semResumo;
        private Resumo resumo;
        private MainScreen ms;
        private final int numCabeleireiras = 5;
        private final int numManicures = 3;
        private final int numDepiladoras = 2;
        private final int numMassagistas = 1;
        private final int numCaixas = 2;
        private final int maxThreadsPermitidas = 1;

        public Salao();
        public Semaphore getsFilasClientes();
        public void setsFilasClientes(Semaphore sFilasClientes);
        public Semaphore getsFilasCaixas();
        public void setsFilasCaixas(Semaphore sFilasCaixas);
}
```

- A clase `Simulador` é a classe que contém o método *main* e tem a responsabilidade de por o todo o sistema para executar.

```
public class Simulador
{
    public static void main(String args[])
    {
                Salao salao = new Salao();
    }
}
```

- As classes `AbstractGUI`, `GerarResumo`, `MainScreen` são as classes que geram a inteface do sistema.

```
public abstract class AbstractGUI extends JFrame
{
        private static final long serialVersionUID = 1L;
        protected int largura;
        protected int altura;

        public AbstractGUI(String titulo, int largura, int altura);
        public int getLargura();
        public void setLargura(int largura);
        public int getAltura();
        public void setAltura(int altura);
}
```

```
1       public class GerarResumoGUI extends AbstractGUI
2       {
3           private static final long serialVersionUID = 1L;
4           private JList<String> linhas;
5               private DefaultListModel<String> listModel;
6
7           public GerarResumoGUI();
8       }
```

```
1       public class MainScreen extends AbstractGUI implements ActionListener
2       {
3               private static final long serialVersionUID = 1L;
4               private List<JList<String>> queueLists;
5               private List<DefaultListModel<String>> listModels;
6               private JButton logButton;
7               private FilasClientes filas;
8               private List<Funcionario> funcionarios;
9               private Semaphore sFilasClientes;
10              private Semaphore sFilasCaixas;
11              private int colunas = 0;
12
13              public MainScreen(FilasClientes filas, List<Funcionario>
                    funcionarios, Semaphore sFilasClientes, Semaphore sFilasCaixas);
14              public void actionPerformed(ActionEvent event);
15              public void criaFilasDeServicos();
16              public void criaFilasCaixas();
17              public void criaColunaFuncionarios();
18              public void atualizaFilasDeServicos();
19              private void atualizaFilaDeServicos(ArrayList<Cliente> fila, int i);
20              public void atualizaFilasCaixas();
21              private void atualizaFilaCaixa(ArrayList<Cliente> fila);
22              public void atualizaFuncionarios();
23      }
```

Na Figura 1 apresentamos o diagrama de classe simplificado da modelagem do sistema e nele podemos ver como é feita a comunicação entre as classes.

3.2 Implementação

Apresentaremos agora como foi desenvolvida a implementação do sistema explicando de forma detalhada cada uma das regras de negócio.

A descrição do projetos nos deu um número fixo de funcionários que devem ser implementados: 5 cabeleireiras, 3 manicures, 2 depiladoras, 1 massagista e 2 caixas. Na nossa implementação, esses funcionários devem acessar filas de atendimentos com determinadas prioridades, essas filas caracterizam a região crítica do sistema e é necessário que exista

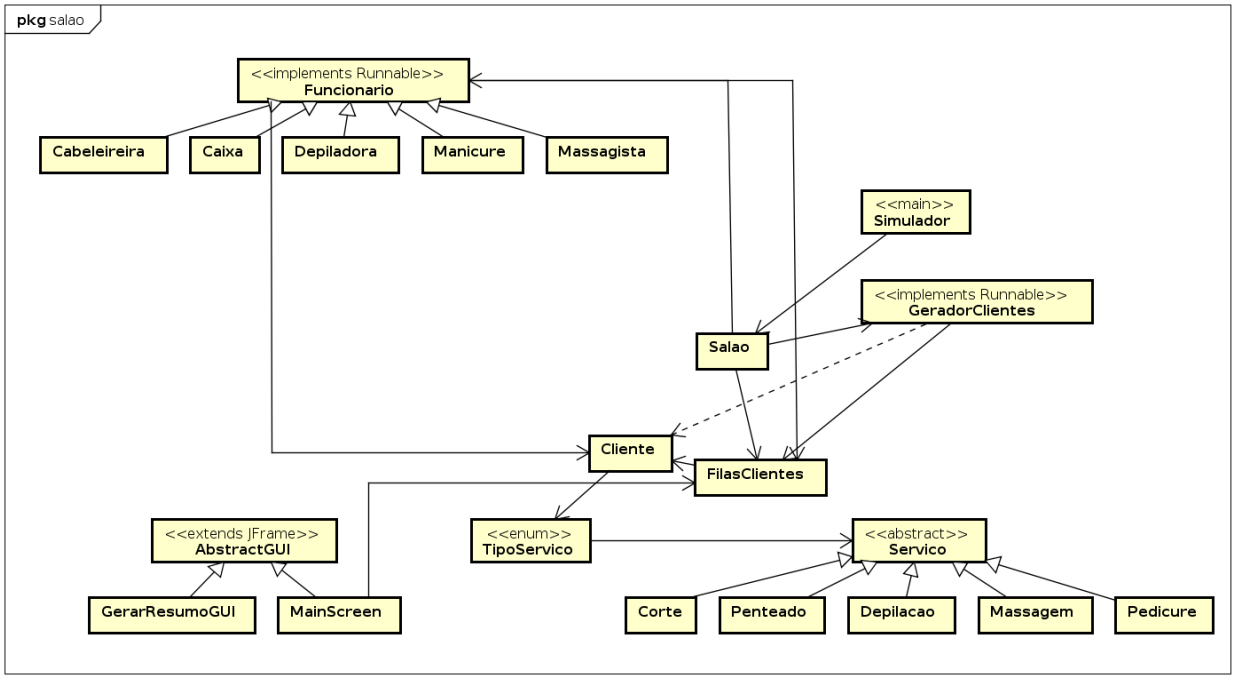

Figura 1: Diagrama de classes do projeto.

algum mecanismo para garantir a exclusão mútua nessa região. Para garantir essa exclusão mútua nós usamos o conceito de semáforo, que já é oferecido pela API da linguagem Java.

Abaixo apresentamos mais detalhes da implementação:

a) A chegada de clientes deve ser simulada segundo um critério aleatório de tempo de chegada entre um e outro variando de 1 a 5 unidades de tempo;

   Para esse caso, a criação dos clientes é feita por uma *thread* que executa uma instância da classe `GeradorClientes`, e que está em execução durante todo o funcionamento do sistema. Essa *thread* gera clientes aleatóriamente em um intervalo de tempo entre 1 e 5 segundos. Esse intervalo de tempo aleatório é gerado pelo método `nexInt()` da classe `Random` oferecida pela biblioteca da linguagem Java. O código abaixo apresenta o método `geraTempoServicos(int quantidade)` da classe `GeradorClientes` que é responsável por essa tarefa.

```
1     // Recebe como parametro a quantidade de tempos que deve ser gerado.
2     // Essa quantidade representa a quantidade de servics que o cliente solicitou.
3     public ArrayList<Integer> geraTempoServicos(int quantidade) {
4               int tempos[] = new int[quantidade];
5               ArrayList<Integer> t = new ArrayList<Integer>();
```

```
6
7               // Gera os tempos aleatoriamente
8               for(int i = 0; i < quantidade; i++) {
9                       tempos[i] = (rand.nextInt(10)+1);
10              }
11
12              Arrays.sort(tempos);
13
14              // Insere de forma decrescente os elemento no ArrayList
15              for(int i = tempos.length-1; i >= 0; i--) {
16                      t.add(tempos[i]);
17              }
18
19              return t;
20      }
```

b) Os clientes devem ser atendidos na ordem de chegada e da disponibilidade dos serviços;

Foram implementadas 6 filas, sendo 5 para atendimento a serviços específicos e 1 para o atendimento aos caixas. Todas as filas obedecem a ordem de chegada e a disponibilidade dos serviços oferecidos. Cada funcionário deve se direcionar as filas para ver se tem algum cliente que tenha solicitado algum serviço que ele oferece. Os semáforos são usados nas filas para garantir que apenas um funcionário acesse a fila por vez. Ao todo são dois semáforos, um para fila de clientes que esperam para ir aos caixas e outo para as filas de clientes que esperam por algum funcionário que executa algum serviço que pode ser solicitado. Abaixo apresentamos os métodos da classe `FilasClientes` que são responsáveis por pegar na fila clientes cujos próximos serviços são os que um determinado funcionário executa.

```
1       // Metodo chamado pela classe Cabeleireira
2       public Cliente getProxParaPenteado() {
3               for(int i = 0; i < filasClientes.size(); i++) {
4                       for(int j = 0; j < filasClientes.get(i).size(); j++) {
5                               if(filasClientes.get(i).get(j).proximoServico().getTipo()
                                    == TipoServico.PENTEADO) {
6                                       Cliente c = filasClientes.get(i).get(j);
7                                       removeDeFilaClientes(i, c);
8                                       return c;
9                               }
10                      }
11              }
12              return null;
13      }
14
15      // Metodo chamado pela classe Cabeleireira
```

```
16          public Cliente getProxParaCorte() {
17                  for(int i = 0; i < filasClientes.size(); i++) {
18                          for(int j = 0; j < filasClientes.get(i).size(); j++) {
19                                  if(filasClientes.get(i).get(j).proximoServico().getTipo()
                                        == TipoServico.CORTE) {
20                                      Cliente c = filasClientes.get(i).get(j);
21                                      removeDeFilaClientes(i, c);
22                                      return c;
23                                  }
24                          }
25                  }
26                  return null;
27          }
28
29          // Metodo chamado pela classe Depiladora
30          public Cliente getProxParaDepilacao() {
31                  for(int i = 0; i < filasClientes.size(); i++) {
32                          for(int j = 0; j < filasClientes.get(i).size(); j++) {
33                                  if(filasClientes.get(i).get(j).proximoServico().getTipo()
                                        == TipoServico.DEPILACAO) {
34                                      Cliente c = filasClientes.get(i).get(j);
35                                      removeDeFilaClientes(i, c);
36                                      return c;
37                                  }
38                          }
39                  }
40                  return null;
41          }
42
43          // Metodo chamado pela classe Manicure
44          public Cliente getProxParaPedicure() {
45                  for(int i = 0; i < filasClientes.size(); i++) {
46                          for(int j = 0; j < filasClientes.get(i).size(); j++) {
47                                  if(filasClientes.get(i).get(j).proximoServico().getTipo()
                                        == TipoServico.PEDICURE) {
48                                      Cliente c = filasClientes.get(i).get(j);
49                                      removeDeFilaClientes(i, c);
50                                      return c;
51                                  }
52                          }
53                  }
54                  return null;
55          }
56
57          // Metodo chamado pela classe Massagista
58          public Cliente getProxParaMassagem() {
59                  for(int i = 0; i < filasClientes.size(); i++) {
60                          for(int j = 0; j < filasClientes.get(i).size(); j++) {
61                                  if(filasClientes.get(i).get(j).proximoServico().getTipo()
                                        == TipoServico.MASSAGEM) {
62                                      Cliente c = filasClientes.get(i).get(j);
63                                      removeDeFilaClientes(i, c);
64                                      return c;
65                                  }
66                          }
67                  }
68                  return null;
69          }
70
71          // Metodo chamado pela classe Caixa
72          public Cliente getProxParaCaixa() {
```

```
73              for(int i = 0; i < filasCaixas.size(); i++) {
74                      for(int j = 0; j < filasCaixas.get(i).size();) {
75                              Cliente c = filasCaixas.get(i).get(j);
76                              removeDeFilaCaixas(c);
77                              return c;
78                      }
79              }
80              return null;
81      }
```

c) A cabeleireira alocada para realizar um corte também lava o cabelo do cliente;

Como não distinguimos qual serviço um determinado funcionário está executando em um determinando momento, não tratamos dessa regra, até pelo falo de um cliente não poder escolher o serviço de lavagem, como é especificados mais abaixo no item j. Abaixo mostramos o trecho de código onde uma cabeleireira executa o atendimento a um cliente.

```
1       public void run() {
2               Cliente c[] = new Cliente[2];
3               while(true) {
4                       c[0] = null;
5                       c[1] = null;
6
7                       try {
8                               this.sFilasClientes.acquire();
9                               c[0] = this.filas.getProxParaCorte();
10                              c[1] = this.filas.getProxParaPenteado();
11                      } catch (InterruptedException e) {
12                              // TODO Auto-generated catch block
13                              e.printStackTrace();
14                      } finally {
15                              this.sFilasClientes.release();
16                      }
17
18              /*
19               * A sequencia se if-esle abaixo verifica se um clientes solicitou
20               * so core, so penteado, e corte e penteado juntos conforme
21               * especificado no item h.
22               */
23
24              // Verifica se o cliente solicitou so penteado
25                      if(c[0] != null && c[1] == null) {
26                              this.cliente = c[0];
27                              c[0].setFuncionario(this);
28
29                              //Execucao feira por uma cabelereira
30                              System.out.println(Thread.currentThread().getName() + ":
                                    Atendendo cliente" + c[0].getId());
31                              try {
```

```
32                              Thread.sleep(1000*c[0].proximoServico().getTempo());
33                      }
34                      catch(InterruptedException ex) {
35                              Thread.currentThread().interrupt();
36                      }
37                      c[0].popServico();
38                      this.reposicionaCliente();
39                      c[0].setFuncionario(null);
40                      this.cliente = null;
41              }
42
43               // Verifica se o cliente solicitou so corte
44              else if(c[0] == null && c[1] != null) {
45                      this.cliente = c[1];
46                      c[1].setFuncionario(this);
47
48                      //Execucao feira por uma cabelereira
49                      System.out.println(Thread.currentThread().getName() + ":
                            Atendendo cliente" + c[1].getId());
50                      try {
51                              Thread.sleep(1000*c[1].proximoServico().getTempo());
52                      }
53                      catch(InterruptedException ex) {
54                              Thread.currentThread().interrupt();
55                      }
56                      c[1].popServico();
57                      this.reposicionaCliente();
58                      c[1].setFuncionario(null);
59                      this.cliente = null;
60              }
61               // Verifica se o cliente solicitou corte e penteado
62              else if(c[0] != null && c[1] != null) {
63                      if(numeroDaFila(c[0]) > numeroDaFila(c[1])) {
64                              this.cliente = c[0];
65                              c[0].setFuncionario(this);
66
67                              //Execucao feira por uma cabelereira
68                              System.out.println(Thread.currentThread().getName()
                                    + ": Atendendo cliente" + c[0].getId());
69                              try {
70                                      Thread.sleep(1000*c[0].proximoServico().getTempo());
71                              }
72                              catch(InterruptedException ex) {
73                                      Thread.currentThread().interrupt();
74                              }
75                              c[0].popServico();
76                              this.reposicionaCliente();
77                              c[0].setFuncionario(null);
78                              this.cliente = null;
79                      } else if(numeroDaFila(c[0]) < numeroDaFila(c[1])) {
80                              this.cliente = c[1];
81                              c[1].setFuncionario(this);
82
83                              //Execucao feira por uma cabelereira
84                              System.out.println(Thread.currentThread().getName()
                                    + ": Atendendo cliente" + c[1].getId());
85                              try {
86                                      Thread.sleep(1000*c[1].proximoServico().getTempo());
87                              }
88                              catch(InterruptedException ex) {
89                                      Thread.currentThread().interrupt();
```

```
                                        }
                                        c[1].popServico();
                                        this.reposicionaCliente();
                                        c[1].setFuncionario(null);
                                        this.cliente = null;
                                } else {
                                        Random rand = new Random();
                                        int next = rand.nextInt(2);
                                        this.cliente = c[next];
                                        c[next].setFuncionario(this);

                                        //Execucao feira por uma cabelereira
                                        System.out.println(Thread.currentThread().getName()
                                            + ": Atendendo cliente" + c[next].getId());
                                        try {
                                                Thread.sleep(1000*c[next].proximoServico().getTempo());
                                        }
                                        catch(InterruptedException ex) {
                                                Thread.currentThread().interrupt();
                                        }
                                        c[next].popServico();
                                        this.reposicionaCliente();
                                        c[next].setFuncionario(null);
                                        this.cliente = null;
                                }
                        }
                        try {
                                Thread.sleep(2000);
                        }
                        catch(InterruptedException ex) {
                                Thread.currentThread().interrupt();
                        }
                }
        }
```

d) Cada cliente pode desejar de 1 a todos os serviços oferecidos pelo salão;

A escolha da quantidade de serviços que será alocada a um cliente é feita de forma aleatório também usando o método *nexInt()* da classe `Random`. Abaixo apresentamos o método da classe `GeradorClientes` onde é feita a geração dos serviços.

```
    // Metodo de cria um cliente para o sistema
    public Cliente criaCliente() {
                contClientes += 1;
                boolean flag = false;
                Cliente cliente = new Cliente(contClientes);
                ArrayList<Integer> inserido = new ArrayList<Integer>(); // evita a
                    repeticao de tipos de servicos
                int quantServicos = 0;

                // O trecho abaixo gera a quantidade de servicos de forma aleatoria
```

```
10              int porcentagemQtd = rand.nextInt(100)+1;
11
12              if(porcentagemQtd >= 1 && porcentagemQtd <= 30)
                    // 30% dos clientes desejam todos os servicos
13              {
14                  quantServicos = 5;
15              }
16              else if(porcentagemQtd >= 31 && porcentagemQtd <= 65)            //35%
                    desejam 4
17              {
18                  quantServicos = 4;
19              }
20              else if(porcentagemQtd >= 66 && porcentagemQtd <= 85)            // 20%
                    desejam 3
21              {
22                  quantServicos = 3;
23              }
24              else if(porcentagemQtd >= 86 && porcentagemQtd <= 95)             //
                    10% apenas 2
25              {
26                  quantServicos = 2;
27              }
28              else if(porcentagemQtd >= 96 && porcentagemQtd <= 100)
                    // 5% apenas 1
29              {
30                  quantServicos = 1;
31              }
32
33              // Faz a insercao da escolha dos clientes inserindo por ordem de escolha
34              // Nao deixa escolher mais de uma vez um mesmo servico
35              for(int i = 0; i < quantServicos; i++)
36              {
37                  // Vamos assumir que o maximo aqui eh 155%
38                  int tipoServico = 0;
39                  int porcentagemTipo = rand.nextInt(100)+1;
40
41                  if(porcentagemTipo >= 1 && porcentagemTipo <= 50)
                        // 50% para corte
42                  {
43                      tipoServico = 1;
44                  }
45                  else if(porcentagemTipo >= 51 && porcentagemTipo <= 90)   // 40%
                        para penteado
46                  {
47                      tipoServico = 2;
48                  }
49                  else if(porcentagemTipo >= 91 && porcentagemTipo <= 120)  // 30%
                        para pedicure
50                  {
51                      tipoServico = 3;
52                  }
53                  else if(porcentagemTipo >= 121 && porcentagemTipo <= 140)  //
                        20% para depilacao
54                  {
55                      tipoServico = 4;
56                  }
57                  else if(porcentagemTipo >= 141 && porcentagemTipo <= 155)  //
                        15% para massagem
58                  {
59                      tipoServico = 5;
60                  }
```

```
61
62                      if(inserido.isEmpty() == false)
63                      {
64                              // Enquanto for servico repetido, gera outro
65                              // (Na realidade, se gerar um igual ele incrementa o
                                    valor e testa novamente)
66                              // Fiz assim pra simplificar
67                              while(flag == false)
68                              {
69                                      for(int num : inserido)
70                                      {
71                                              if(num == tipoServico)
72                                              {
73                                                      //tipoServico = rand.nextInt(5)+1;
74                                                      if(tipoServico < 5)
75                                                      {
76                                                              tipoServico++;
77                                                      }
78                                                      else
79                                                      {
80                                                              tipoServico = 1;
81                                                      }
82
83                                                      flag = true;
84                                                      break;
85                                              }
86                                      }
87
88                                      if(flag == true)
89                                      {
90                                              flag = false;
91                                      }
92                                      else
93                                      {
94                                              flag = true;
95                                      }
96                              }
97                      }
98
99                      flag = false;
100                     inserido.add(tipoServico);
101                     switch(tipoServico)
102                     {
103                     case 1:
104                             Servico servico = new Corte(0);
105                             cliente.incluirServico(servico);
106                             break;
107                     case 2:
108                             servico = new Penteado(0);
109                             cliente.incluirServico(servico);
110                             break;
111                     case 3:
112                             servico = new Pedicure(0);
113                             cliente.incluirServico(servico);
114                             break;
115                     case 4:
116                             servico = new Depilacao(0);
117                             cliente.incluirServico(servico);
118                             break;
119                     case 5:
120                             servico = new Massagem(0);
```

```
121                             cliente.incluirServico(servico);
122                             break;
123                     default:
124                             throw(new IndexOutOfBoundsException("Opcao nao
                                    existe!!!"));
125                     }
126             }
127             return cliente;
128     }
```

e) Um cliente não deve prender outro que esteja atrás de si e que deseja um serviço que esteja disponível;

Na implementação da fila fazemos com que os clientes sejam atendidos baseado na disponibilidade do funcionário, neste caso o cliente fica na fila, porém se todos os outros clientes que estão na frente dele não tem como próximo serviço o serviço que ele tem como próximo e existe um profissional apto para atendê-lo, então ele é atendido. O método apresentado no item *c* apresenta com foi implementada essa regra de negócio.

f) Todo corte deve ser sempre precedido de uma lavagem;

Como não distinguimos qual serviço um determinado funcionário está executando em um determinando momento, não tratamos dessa regra, até pelo falo que um cliente não poder escolher o serviço de lavagem, como é especificados mais abaixo no item *j*. O método apresentado no item *c* apresenta com foi implementada as regras de negócio que envolvem os atendimentos feito por cabeleireiras.

g) O tempo gasto em cada serviço por cada cliente deve ser gerado aleatoriamente considerando a seguinte ordem decrescente de duração: penteado, corte, depilação, pés e mãos, massagem e lavagem;

A geração o tempo de serviço é feita de forma aleatória e seguindo o critério de atribuição de tempo descrito nessa regra de negócio. O código apresentado no item *a* mostra como é feita a geração aleatória dos tempos e o código abaixo apresenta como é feita a tribuição de tempo a cada serviço solicitado por um determinado cliente.

```
1     public void run() {
2               while(true) {
3                       Cliente c = criaCliente();
4                       ArrayList<Integer> t = new ArrayList<Integer>();
5                       t = geraTempoServicos(c.getServicosSolicitados().size());
6
7                       // Atribui o tempo aos servicos da forma especificada na
                            descricao do projeto
8                       for(Servico s: c.getServicosSolicitados()) {
9                               if(s.getTipo() == TipoServico.PENTEADO) {
10                                      s.setTempo(t.get(0));
11                                      t.remove(0);
12                              } else if(s.getTipo() == TipoServico.CORTE) {
13                                      s.setTempo(t.get(0));
14                                      t.remove(0);
15                              } else if(s.getTipo() == TipoServico.DEPILACAO) {
16                                      s.setTempo(t.get(0));
17                                      t.remove(0);
18                              } else if(s.getTipo() == TipoServico.PEDICURE) {
19                                      s.setTempo(t.get(0));
20                                      t.remove(0);
21                              } else if(s.getTipo() == TipoServico.MASSAGEM) {
22                                      s.setTempo(t.get(0));
23                                      t.remove(0);
24                              }
25                      }
26
27                      filas.insereEmFilaClientes(0, c);
28
29                      try {
30                              // Tempo de geracao de clientes: 1 - 5 segundos
31                              Thread.sleep(1000*(rand.nextInt(5)+1));
32                      } catch(InterruptedException ex) {
33                               Thread.currentThread().interrupt();
34                      }
35              }
36      }
```

h) O preço de cada serviço é de 50 reais para penteado, 30 reais para corte, 40 reais para corte e penteado, 0 reais para lavagem, 30 reais para pedicure, 40 reais para depilação e 20 reais para massagem;

   Para esse caso, o valor especificado acima é cobrado assim que o cliente é atendido pelo caixa e esse valor é usado para fazer o cálculo dos faturamentosma do salão.

i) Em geral 30% dos clientes desejam todos os serviços, 35% desejam 4, 20% desejam 3, 10% apenas 2 e 5% apenas 1;

   Para a quantidade de serviços geramos números aleatórios em intervalos específicos para fazer com que a porcentagem seja dada pela

geração desses números. Depois disso, atribuímos a quantidade de serviços fixa para aquele cliente. O código apresentado no item *c* apresenta como é gerada a quantidade de serviços solicitados por um determinado cliente.

j) Os serviços também são procurados segundo um percentual médio de 50% para corte, 40% para penteado, 30% para pedicure, 20% para depilação, 15% para massagem;

   Para esse caso também geramos números aleatórios em intervalos específicos para fazer com que a porcentagem seja dada pela geração desse número. Depois da geração, adicionamos o serviço na lista de serviços do cliente. O código apresentado no item *c* apresenta como é gerada a escolha dos serviços para os clientes.

k) A política adotada pelo dono do estabelecimento é que cada profissional recebe 40% do total faturado por ele durante o dia de trabalho;

   O cálculo do faturamento do cliente é mostrado quando o usuário solicita o resumo clicando no botão da interface gráfica do sistema.

l) O salão tem por regra de negócio otimizar o tempo do cliente, atendendo-o da melhor forma e no menor tempo possível;

   As filas implementadas tem as seguintes características: a fila 1 é para os clientes que acabaram de chegar no salão, fila 2 pra os clientes que já foram atendidos 1 vez a ainda tem serviço para ser atendido, fila 3 clientes que foram atendidos 2 vezes a ainda tem serviço para ser atendido, fila 4 para clientes que já foram atendidos 3 vezes e ainda tem serviços para ser atendidos e a fila 5 é para um cliente no seu ultimo serviço. A passagem de uma fila para outra é feira pela *thread* que vai executar o serviço de um determinado cliente. Ela recebe a instância do cliente e da próxima fila que ele será inserido, caso ele não possua mais serviços a serem atendidos, então é inserido na fila dos caixas.

   A politica de prioridade de atendimento e atribuição nas filas é pensada de forma que o cliente com mais serviços sejam atendidos da forma mais rápida possível, com isso quando um cliente é atendido ele é inserido numa fila com maior prioridade, ou seja, uma fila de maior índice, já que a ordem de prioridade das filas é dada de forma

decrescente sendo a fila 5 a de maior prioridade e a 1 a de menor prioridade.

m) O sistema deve apresentar um resumo do movimento e do faturamento realizado;

   O resumo do movimento e faturamento é gerado a partir do momento que o usuário clica no botão da interface gráfica. As informações apresentadas são: faturamento de todos os funcionários, total de atendimentos registrados em todos os funcionários, total em dinheiro registrado por todos os funcionários, total de atendimentos registrados pelos caixas, total em dinheiro recebido pelos caixas, atendimentos que ainda faltam ser computados pelo caixa, dinheiro que ainda será recebido pelo caixa e a quantidade de clientes que ainda fata ir ao caixa.

n) Construir uma representação na tela do monitor da movimentação nas filas de entrada, de espera por cada profissional;

   Tal interface gráfica foi implementada e apresenta as filas de clientes citadas anteriormente e possibilita o usuário do sistema obter um resumo das movimentações através de um botão. A interface pode ser vista na seção 4.

4 Resultados computacionais

A implementação funcionou bem e durante várias horas, isso pode ser observado nas imagens a seguir. A interface gráfica garante uma boa visualização do funcionamento do sistema. Além disso, o paradigma utilizado, juntamente com a modelagem proposta, garante extensibilidade. Por exemplo, o sistema pode-se adequar tranquilamente a uma nova quantidade de funcionários ou até a inclusão de outros funcionários.

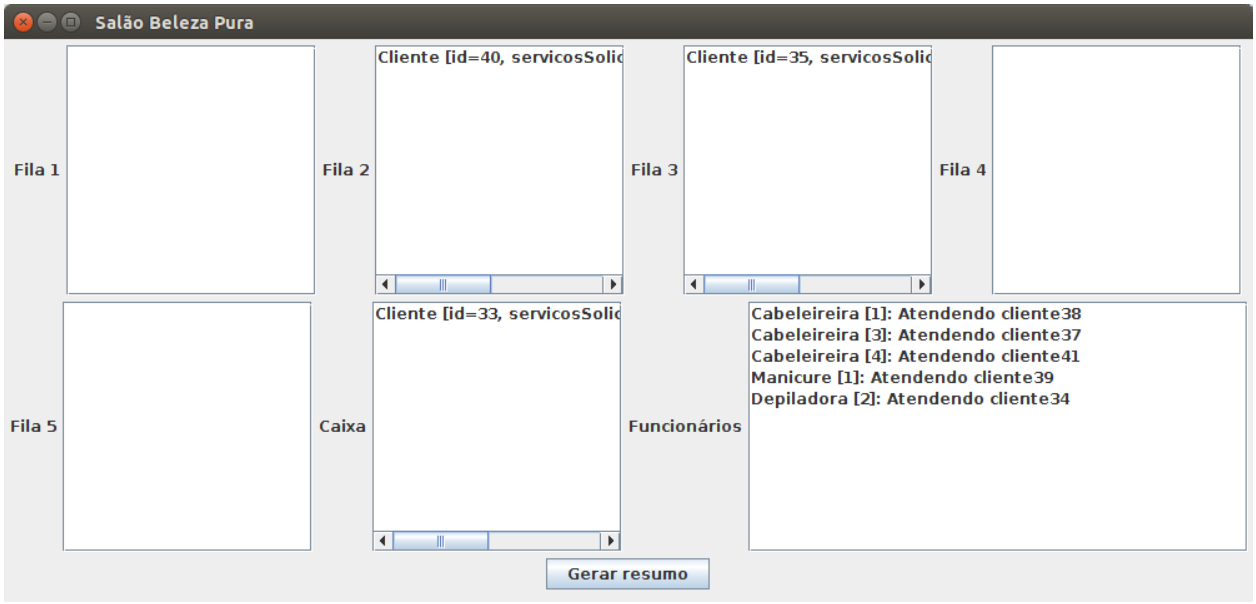

Figura 2: Tela principal que apresenta filas de clientes e funcionários ativos.

No projeto consta um arquivo `readme` que contém as instruções para executar o simulador.

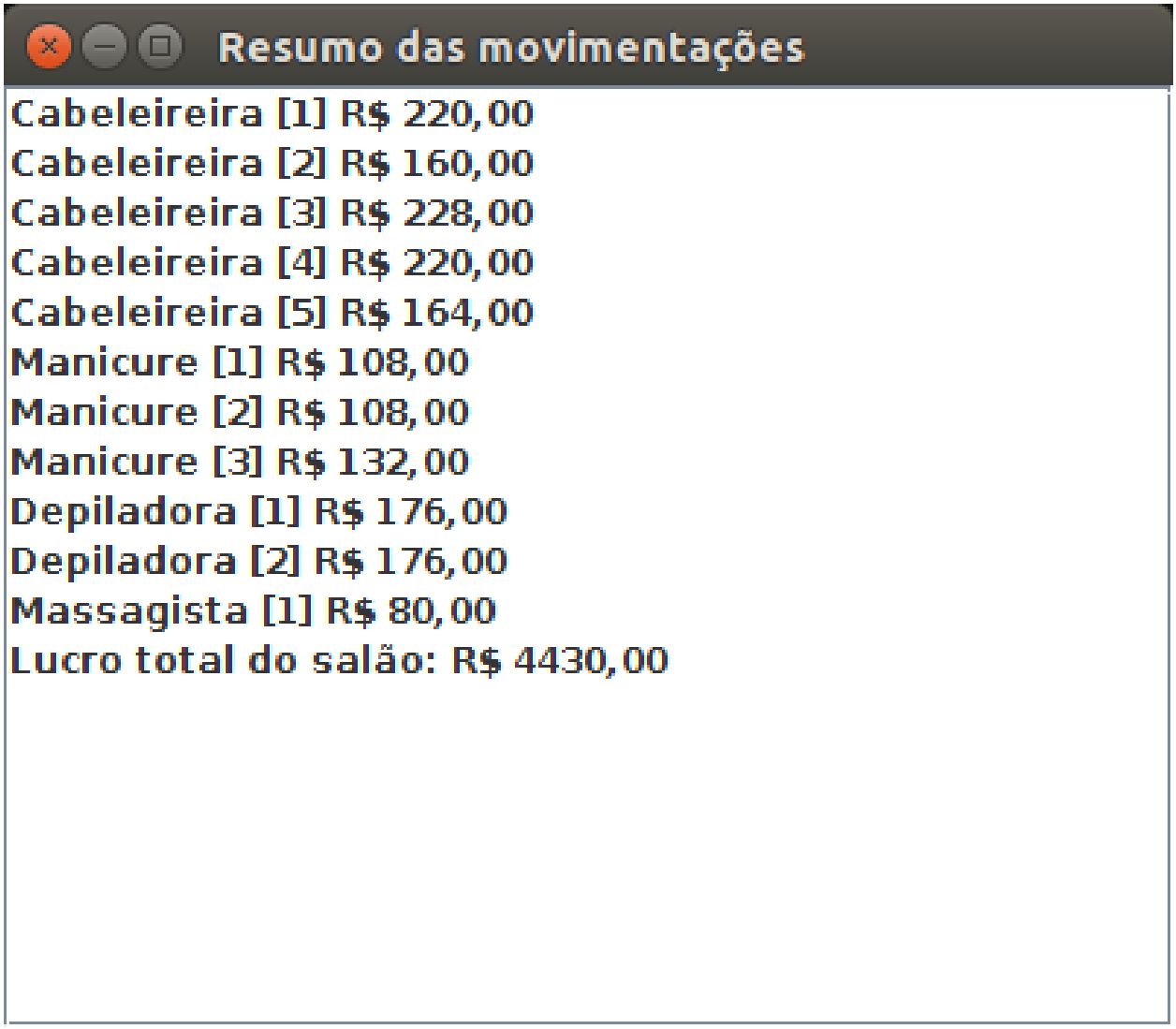

Figura 3: Tela secundária que mostra resumo das movimentações no salão.

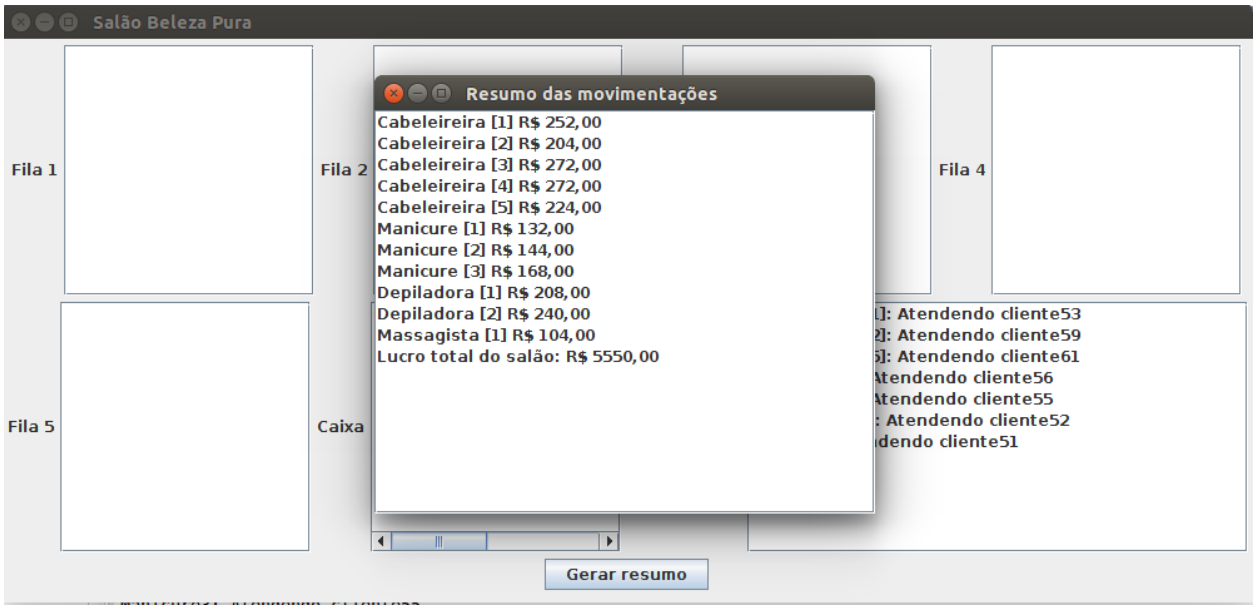

Figura 4: As duas telas são independentes entre si.