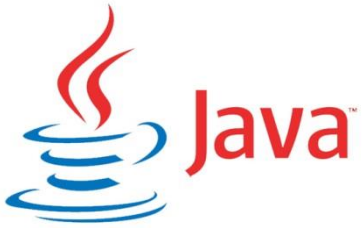
Lógica de Programação

Unidade 20 – ArrayList: Operações de Busca



QI ESCOLAS E FACULDADES
Curso Técnico em Informática

SUMÁRIO

INTRODUÇÃO

Nesta unidade iremos aprender as operações de busca dentro de uma lista.

TIPOS DE BUSCAS

Quando trabalhamos com coleções de objetos é muito comum termos a necessidade de buscar algo dentro da coleção, essa busca pode ser **específica (procurando um único elemento)** ou **abrangente (procurando um subconjunto de elementos)**.

Busca Específica

Uma busca específica é quando o critério utilizado para a busca é **exclusivo** para cada objeto. Exemplo: busca por RG, CPF, placa do veículo, nome completo, e assim por diante. Neste tipo de busca teremos apenas um retorno, um objeto como resultado, afinal não temos dois clientes com o mesmo RG, ou dois carros com a mesma placa, isto é algo específico de cada objeto. Nas buscas específicas o resultado será apenas um objeto.

Portanto, neste tipo de busca podemos ter nenhum ou um único objeto como retorno.

Busca Abrangente

Uma busca abrangente é quando o critério da busca pode resultar em **vários objetos**. Exemplo: buscar todas as pessoas do sexo feminino, todos os alunos de lógica em EaD, os clientes aniversariantes do dia. Na busca abrangente teremos como retorno uma lista de objetos.

Neste tipo de busca podemos ter nenhum, um ou vários objetos como resultado.

Procedimento de Busca

Para programar uma busca, precisamos de um laço de repetição para percorrer cada elemento (cadastro) da lista, analisando os dados que atendem ao critério da busca.

Exemplos de Buscas

Exemplo de busca específica

Vamos pensar na nossa lista de compras, queremos encontrar um produto específico, imagine que na lista há apenas um nome para cada produto, neste caso não teremos dois produtos iguais, certo? Portanto faremos uma busca pelo nome do produto.

No diagrama, acrescentaríamos o método da seguinte forma:

+pesquisarProdutoPorNome(nome:String):Produto

Abaixo temos a sintaxe do método de pesquisa. Repare que na linha 37 temos um laço de repetição no qual a variável **i** representa o índice que começa em 0 e vai até o final da lista ou até encontrar o produto desejado.

Na linha 38 temos uma condicional comparando o nome do produto que está na lista com o nome que o usuário pediu para localizar. Observe que utilizamos o método "*equalsIgnoreCase*". É um método da classe String e serve para comparar uma *String* com outra, sem diferenciar maiúsculas de minúsculas.

```
35      public Produto pesquisarProdutoPorNome(String nome){
36          Produto p1 = null;
37          for(int i = 0; i<this.lista.size() && p1==null; i++){
38              if(this.lista.get(i).getNome().equalsIgnoreCase(nome)){
39                  p1 = this.lista.get(i);
40              }
41          }
42          return p1;
43      }
44  }
```

Figura 1 – Exemplo de código

Este primeiro exemplo é uma busca específica, pois especificamos um item que será comum a apenas um objeto que está na lista, portanto o retorno será apenas um elemento ou nenhum, caso não localize o produto com aquele nome.

Exemplo de busca abrangente

Novamente com o nosso projeto lista de compras iremos pesquisar agora os produtos que tiverem um valor unitário de até R$ 10,00. Pense, neste caso, podemos ter nenhum, um ou vários produtos que custam até R$ 10,00.

+pesquisarProdutosAte10Reais():ArrayList<Produto>

Neste método não precisaremos de argumentos, pois o usuário não precisará informar nenhum dado para a pesquisa.

```
19  public ArrayList<Produto> pesquisarProdutosAte10Reais(){
20      ArrayList<Produto> produtosAte10 = new ArrayList<>();
21      for(int i=0; i<this.lista.size(); i++){
22          if(this.lista.get(i).getValorUnitario() <= 10){
23              produtosAte10.add(this.lista.get(i));
24          }
25      }
26      return produtosAte10;
27  }
```

Figura 2 – exemplo de código

Observe a figura 2, note que no método pesquisarProdutosAte10Reais não temos argumento e será um método retornando uma lista de produtos. Isso por que podemos ter vários produtos custando até 10 reais.

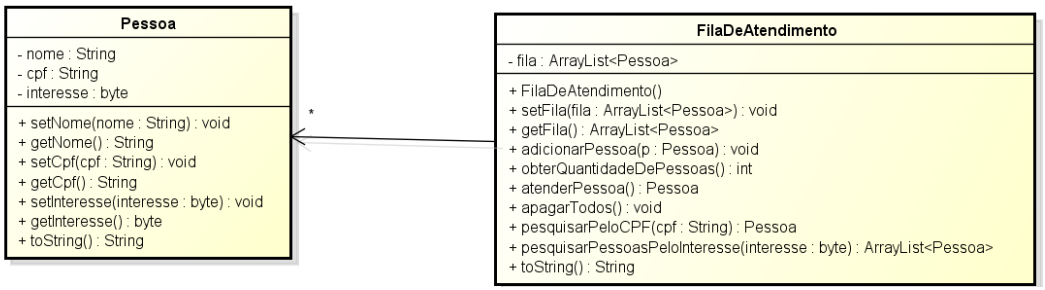
Na linha 21 temos um laço de repetição no qual irá percorrer toda a lista de elemento a elemento até o final da mesma.

Na linha 22 temos uma condicional para testar se e o elemento da lista possui o *valorUnitario* menor ou igual a 10 reais, se tiver, irá armazenar o mesmo em uma outra lista criada para guardar os elementos que possuírem valores até 10 reais, esta lista será retornada ao final do método.

Exemplo Completo

Vamos pensar em uma fila de atendimento bancário, nesta fila cada pessoa que chega especifica seu nome, CPF e seu interesse (Os interesses podem ser: 1-pagamento de conta, 2-Recebimento de salário, 3-Outros).

Vamos desenvolver um sistema para esta fila de atendimento, seu sistema deve permitir **adicionar** pessoas na fila de atendimento, **atender** uma pessoa (chame pelo nome e exclua a mesma automaticamente da fila), **listar** todas as pessoas que estão na fila, ver a **quantidade** de pessoas que estão na fila, **pesquisar** uma pessoa pelo CPF, **listar** as pessoas de acordo com interesse informado no sistema (exemplo: pessoas que desejam efetuar pagamento, recebimento ou outro interesse). No final do expediente tenha uma forma de **excluir** todas as pessoas da fila. Temos, assim, um objeto Pessoa e um objeto FilaDeAtendimento, de modo que uma fila de atendimento contém uma coleção de objetos da classe Pessoa.



Classe Pessoa

```java
public class Pessoa {
    private String nome;
    private String cpf;
    private byte interesse;

    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
    public byte getInteresse() {
        return interesse;
    }
    public void setInteresse(byte interesse) {
        this.interesse = interesse;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    @Override
    public String toString() {
        String texto;
        switch(this.interesse){
            case 1:
                texto = "Pagamento de conta";
                break;
            case 2:
                texto = "Recebimento de salário";
                break;
            default:
                texto = "Outro";
        }
        return "\nNome: " +  nome + ", cpf: " + cpf + ", interesse: " + texto + '}';
    }
}
```

Figura 3 – Classe Pessoa

A classe possui os atributos nome, CPF e interesse privados, os métodos assessores e modificadores *sets*/*ges* e o método *toString* para retornar os dados da Pessoa.

Classe FilaDeAtendimento

```
import java.util.ArrayList;
public class FilaDeAtendimento {
    private ArrayList <Pessoa> fila;

    public FilaDeAtendimento(){
        this.fila = new ArrayList<>();
    }
    public ArrayList<Pessoa> getFila() {
        return fila;
    }
    public void setFila(ArrayList<Pessoa> fila) {
        this.fila = fila;
    }
    public void adicionarPessoa(Pessoa p){
        this.fila.add(p);
    }
    public int obterQuantidadeDePessoas(){
        return this.fila.size();
    }
    public Pessoa atenderPessoa(){
        Pessoa p1 = this.fila.get(0);
        this.fila.remove(0);
        return p1;
    }
    public void apagarTodos(){
        this.fila.clear();
    }
    public Pessoa pesquisarPeloCPF(String cpf){
        Pessoa p2 = null;
        for(int i = 0; i<this.fila.size(); i++){
            if(this.fila.get(i).getCpf().equals(cpf)){
                p2 = this.fila.get(i);
            }
        }
        return p2;
    }
    public ArrayList<Pessoa> pesquisarPessoasPeloInteresse(byte interesse){
        ArrayList<Pessoa> listaInteresse = new ArrayList<>();
        for(int i = 0; i<this.fila.size(); i++){
            if(this.fila.get(i).getInteresse()==interesse){
                listaInteresse.add(this.fila.get(i));
            }
        }
        return listaInteresse;
    }
    @Override
    public String toString() {
        return "Fila de Atendimento\n" + fila;
    }
}

```

Figura 4 – Classe FilaDeAtendimento

A classe FilaDeAtendimento temos um *ArrayList* chamado fila que irá armazenar objetos do tipo Pessoa.

Perceba que o método atenderPessoa() – linha 20 – neste método criamos uma variável p1 do tipo pessoa na qual irá armazenar a primeira pessoa da fila, após removemos a primeira pessoa da fila (simbolizando que a mesma já foi atendida). E retornamos a variável p1, como se estivéssemos chamando a mesma para o atendimento.

Na linha 28 iniciamos um método que nos permite buscar uma pessoa na fila através do seu CPF, um método de pesquisa específica que irá retornar nenhuma ou apenas uma pessoa, caso tenha o CPF na qual está sendo pesquisado.

Na linha 37 temos o método pesquisarPessoasPeloInteresse() no qual o usuário irá informar qual interesse quer pesquisar e o sistema retornará nenhum, uma ou várias pessoas que estiverem com o mesmo interesse, exemplo: visualizar quais as pessoas que desejam pagar conta, quais irão apenas receber salário, ou quais possuem outro interesse qualquer na agência bancária.

Classe Main

Na classe *Main*, apresentaremos ao usuário as seguintes opções:

```
System.out.println("Digite sua opção:");
System.out.println("1 - Adicionar um cliente na fila");
System.out.println("2 - Ver cliente da fila");
System.out.println("3 - Ver quantidade de clientes que estão na fila");
System.out.println("4 - Atender um cliente");
System.out.println("5 - Pesquisar um cliente pelo CPF");
System.out.println("6 - Pesquisar clientes pelo interesse");
System.out.println("7 - Retirar todos os clientes da fila");
System.out.println("0 - Sair");
```

```java
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);
        FilaDeAtendimento f1 = new FilaDeAtendimento();
        byte escolha;
        do{
            System.out.println("Digite sua opcao:");
            System.out.println("1 - Adicionar uma cliente na fila");
            System.out.println("2 - Ver cliente da fila");
            System.out.println("3 - Ver quantidade de clientes que estão na fila");
            System.out.println("4 - Atender um cliente");
            System.out.println("5 - Pesquisar um cliente pelo CPF");
            System.out.println("6 - Pesquisar clientes pelo interesse");
            System.out.println("7 - Retirar todos os clientes da fila");
            System.out.println("0 - Sair");
            escolha = ler.nextByte();
            switch (escolha){
                case 1:
                    Pessoa p1 = new Pessoa();
                    System.out.println("Informe o nome do cliente:");
                    p1.setNome(ler.next());
                    do{
                        System.out.println("Informe o cpf do cliente:");
                        p1.setCpf(ler.next());
                        if(p1.getCpf().length()!=11){
                            System.out.println("Erro, CPF inválido.");
                        }
                    }while(p1.getCpf().length()!=11);
                    do{
                        System.out.println("Informe o interesse do cliente");
                        System.out.println("1 - Pagamento de Conta");
                        System.out.println("2 - Recimento de salário");
                        System.out.println("3 - Outros");
                        p1.setInteresse(ler.nextByte());
                        if(p1.getInteresse()<1 | p1.getInteresse()>3){
                            System.out.println("Erro, inválido.");
                        }
                    }while(p1.getInteresse()<1 | p1.getInteresse()>3);
                    f1.adicionarPessoa(p1);
                    System.out.println("Pessoa adicionada na fila com sucesso.");
                    break;
                case 2:
                    if(f1.getFila().isEmpty()){
                        System.out.println("Não há clientes na fila.");
                    } else {
                        System.out.println(f1);
                    }
                    break;
                case 3:
                    System.out.println("Total de clientes na fila: "
                            +f1.obterQuantidadeDePessoas());
                    break;
```

```
55                  case 4:
56                      if(f1.getFila().isEmpty()){
57                          System.out.println("Não há clientes para atender.");
58                      } else {
59                          System.out.println("Cliente: " +f1.atenderPessoa());
60                      }
61                  break;
62                  case 5:
63                      System.out.println("Informe o CPF do cliente para efetuar a pesquis
64                      String nome = ler.next();
65                      Pessoa resultado = f1.pesquisarPeloCPF(nome);
66                      if(resultado==null){
67                          System.out.println("Não há este cliente na fila.");
68                      } else {
69                          System.out.println(resultado);
70                      }
71                      break;
72                  case 6:
73                      System.out.println("Informe o interesse para listar os clientes:");
74                      System.out.println("1 - Pagamento de Conta");
75                      System.out.println("2 - Recimento de salário");
76                      System.out.println("3 - Outros");
77                      byte pesquisa = ler.nextByte();
78                      ArrayList<Pessoa> lista = f1.listarPessoasPeloInteresse(pesquisa);
79                      if(lista.isEmpty()){
80                          System.out.println("Não há clientes com esse interesse");
81                      } else {
82                          System.out.println(lista);
83                      }
84                      break;
85                  case 7:
86                      if(f1.getFila().isEmpty()){
87                          System.out.println("Nao há clientes na fila.");
88                      } else{
89                          f1.apagarTodos();
90                          System.out.println("Clientes removidos da fila.");
91                      }
92                      break;
93                  case 0:
94                      System.out.println("Sistema encerrado.");
95                      break;
96                  default:
97                      System.out.println("Opcao invalida.");
98                      break;
99              }
100         }while(escolha!=0);
101     }
102 }
```

Figura 5 – Classe Main

Na figura 5 – temos o código da classe *Main*, perceba que a mesma instancia a classe FilaDeAtendimento e dentro do case 1 do menu, instancia a classe Pessoa.

Perceba também a importância de verificar se há dados antes de executar cada operação, exemplo: no *case* 2: antes de atender uma pessoa, verificamos se há pessoas na fila para atender, caso a fila esteja vazia retornará ao usuário – linha 46 – "Não há clientes na fila."

BIBLIOGRAFIA

ORACLE. **ArrayList**. Java Plataform SE 7, 2012. Disponível em http://docs.oracle.com/javase/7/docs/api/. Acesso em 11 Dez. 2012.