Lógica de Programação

Unidade 19 – ArrayList: coleções dinâmicas de objetos

QI ESCOLAS E FACULDADES
Curso Técnico em Informática

SUMÁRIO

INTRODUÇÃO

Nesta unidade estudaremos as estruturas dinâmicas de armazenamento coletivo de objetos, conhecidas como *ArrayList*.

O QUE É UM ARRAYLIST?

Um *ArrayList* é uma coleção dinâmica capaz de armazenar um número indeterminado de objetos.

> *A classe ArrayList permite criar um objeto que é capaz de armazenar e gerenciar uma coleção de outros objetos.*

Um objeto da classe *ArrayList* é semelhante à estrutura *array*, porém os *arrays* são estáticos, nos quais temos que determinar um número de elementos que serão armazenados e esse número de elementos permanece até o final do programa, já a coleção *ArrayList* é dinâmica, isso significa que a mesma não precisa ter um número determinado para armazenamento e sua estrutura se adapta com cada objeto inserido.

VANTAGENS EM UTILIZAR ARRAYLIST

Vantagens em relação à estrutura *array*:

- → Podemos guardar um conjunto de dados especificando o tipo de objeto;
- → É dinâmico: não possui tamanho definido;
- → Já possui métodos para facilitar o gerenciamento da coleção – inserção, exclusão, e assim por diante.

Importação da classe ArrayList

Para criarmos um coleção dinâmica de objetos temos que criar um objeto (atributo) na qual terá uma coleção de outros objetos, essa criação só pode ser feita após a importação da classe *ArrayList*.

```
import java.util.ArrayList;
```

Declaração de um ArrayList

```
visibilidade ArrayList <TipoObjeto> nomeObjeto;
nomeObjeto = new ArrayList<>();

ou

visibilidade ArrayList <TipoObjeto> nomeObjeto = new ArrayList<>();
```

Exemplo:

```
private ArrayList <Cliente> listaDeClientes;
listaDeClientes = new ArrayList<>();

ou

private ArrayList <Cliente> listaDeClientes = new ArrayList<>();
```

Tipos de Objetos

Podemos armazenar objetos do tipo números inteiros, números reais, textos, valores booleanos e objetos de classes criadas no sistema. Vejamos um exemplo de cada.

Tipo inteiro

```
private ArrayList <Integer> listaDeNumeros;
listaDeNumeros = new ArrayList<>();
```

```
private ArrayList <Byte> listaDeNumeros;
listaDeNumeros = new ArrayList<>();
```

Tipo real

```
private ArrayList <Double> listaDeSalarios;
listaDeSalarios = new ArrayList<>();
```

Tipo texto

```
private ArrayList <String> listaDeNomes;
listaDeNomes = new ArrayList<>();
```

Tipo booleano

```
private ArrayList <Boolean> listaDeRespostas;
listaDeRespostas = new ArrayList<>();
```

Outras classes

```
private ArrayList <Funcionario> listaDeFuncionarios;
listaDeFuncionarios = new ArrayList<>();
```

> ***Obs.**: Dependendo da versão do seu JDK, pode haver necessidade de repetir o tipo de dado. Exemplo:*
>
> ```
> private ArrayList <Funcionario>listaDeFuncionarios;
> listaDeFuncionarios = new ArrayList<Funcionario>();
> ```

Métodos da classe ArrayList

Método	Função
add(elemento)	Insere um elemento no *ArrayList*
clear()	Limpa a lista toda (apaga tudo)
isEmpty()	Verifica se a lista está vazia (retorna um *boolean*)
size()	Retorna a quantidade de elementos no *ArrayList*
get(índice)	Retorna o elemento que está armazenado no índice especificado
remove(índice)	Remove o elemento contido no índice especificado
remove(objeto)	Remove o objeto especificado

ESTUDO DE CASO: LISTA DE COMPRAS

Vamos imaginar que precisamos de um software que nos ajude a gerenciar uma lista de compras. Eu cadastro os produtos na lista de compras, onde de cada um eu informo o nome, a quantidade e o valor unitário. Eventualmente, quero saber quantos produtos estão adicionados na lista, visualizar os produtos e até mesmo saber o total em dinheiro.

Representação no Diagrama

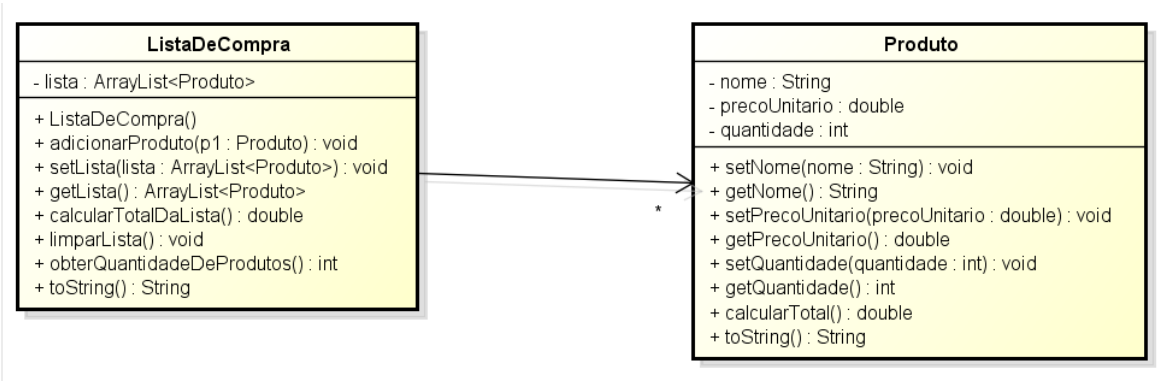

Figura 1- Diagrama de Classe

No diagrama acima temos uma classe **ListaDeCompra** e uma classe **Produto**, a classe ListaDeCompra possui coleção de objetos do tipo Produto, isso significa que cada produto adicionado à lista irá possuir tudo o que foi definido na classe Produto, ou seja, possuirá um nome, uma quantidade e um preço unitário.

Criando a classe Produto

```
1  public class Produto {
2      private String nome;
3      private int quantidade;
4      private double valorUnitario;
5
6      public String getNome() {
7          return nome;
8      }
9      public void setNome(String nome) {
10         this.nome = nome;
11     }
12     public int getQuantidade() {
13         return quantidade;
14     }
15     public void setQuantidade(int quantidade) {
16         this.quantidade = quantidade;
17     }
18     public double getValorUnitario() {
19         return valorUnitario;
20     }
21     public void setValorUnitario(double valorUnitario) {
22         this.valorUnitario = valorUnitario;
23     }
```

```
24      @Override
25      public String toString(){
26          return "\nProduto " +this.nome+ " custa R$ " +this.valorUnitario+
27                  " quantidade " +this.quantidade+ " unidade(s)";
28      }
29      public double calcularTotal(){
30          return this.quantidade * this.valorUnitario;
31      }
32  }
```

Figura 2 - Classe Produto

Perceba que a classe acima, classe Produto, possui apenas os atributos de Produto (itens nos quais iremos cadastrar na lista), os *gets/sets* para os mesmos, o toString (para retornar um produto) e um método calcularTotal no qual irá totalizar o valor de cada produto, multiplicando a quantidade pelo valor unitário.

Criando a classe ListaDeCompra

```
1   import java.util.ArrayList;
2
3   public class ListaDeCompra {
4       private ArrayList<Produto> lista;
5
6       public ListaDeCompra(){
7           lista = new ArrayList<Produto>();
8       }
9       public ArrayList<Produto> getLista() {
10          return lista;
11      }
12      public void setLista(ArrayList<Produto> lista) {
13          this.lista = lista;
14      }
15      @Override
16      public String toString() {
17          return "ListaDeCompras{" + "lista=" + lista + '}';
18      }
19      public void adicionarProduto(Produto p1){
20          this.lista.add(p1);
21      }
22      public void apagarTudo(){
23          this.lista.clear();
24      }
25      public int obterQuantidadeDeProdutos(){
26          return this.lista.size();
27      }
```

```
28      public double calcularTotalDaLista(){
29          double soma = 0;
30          for(int i=0; i<this.lista.size();i++){
31              soma = soma + this.lista.get(i).calcularTotal();
32          }
33          return soma;
34      }
35  }
```

Figura 3 – Classe ListaDeCompra

Note que iniciamos a classe com a importação da classe *ArrayList*. Em seguida, declaramos como atributo desta classe uma coleção de objetos da classe Produto, chamada de *lista*. Em seguida, o construtor instancia o objeto, preparando-o para receber elementos.

Método para adicionar um objeto na lista

Para adicionarmos um objeto na lista utilizamos o método **add()**. Ele sempre adiciona o objeto no final da lista. Observe o código:

```
public void adicionarProduto(Produto p1){
      this.lista.add(p1);
}
```

Este método será um método sem retorno e irá adicionar na lista um objeto produto recebido por passagem de parâmetro.

Método para verificar a quantidade de objetos de uma lista

Para verificar a quantidade de objetos de uma lista utilizamos o método **.size().**

```
public int obterQuantidadeDeProdutos(){
      return this.lista.size();
}
```

Este método é um método com retorno, e irá apenas retornar o total de cadastros.

Método para limpar uma Lista – apagando todos os cadastros

Para excluirmos todos os objetos de uma lista utilizamos o método **.clear().**

```
public void apagarTudo(){
      this.lista.clear();
}
```

Este método é um método sem retorno, ele apenas irá excluir todos os cadastros.

Efetuando operações com os objetos da lista

Na Figura 3, onde temos o código da classe ListaDeCompra, repare que na linha 28 temos um método calcularTotalDaLista(), este método irá acumular o valorTotal de cada produto, portanto temos um laço de repetição que percorre a lista, desde o primeiro produto até o último e temos uma variável acumuladora **soma**, na qual irá somando cada cálculo de total dos produtos. Observe na linha 31:

```
soma = soma + this.lista.get(i).calcularTotal();
```

Note que utilizamos primeiro o método *get* para obter um objeto da classe Produto de dentro da lista, e em seguida solicitamos o método calcularTotal.

Preenchendo a lista com objetos determinados pelo programador

```
ArrayList<Produto> l1 = new Arraylist<>();

Produto p1 = new Produto();
p1.setNome("Bis");
p1.setQuantidade(2);
p1.setValorUnitario(2.99);
l1.adicionarProduto(p1);
```

Na memória:

0	p1[nome: "Bis", quantidade: 2, valorUnitario: 2.99]

l1

Adicionando mais um objeto:

```
Produto p2 = new Produto();
p1.setNome("Sabão");
p1.setQuantidade(10);
p1.setValorUnitario(0.99);
l1.adicionarProduto(p2);
```

Na memória:

0	p1[nome: "Bis", quantidade: 2, valorUnitario: 2.99]
1	p2[nome: "Sabão", quantidade: 10, valorUnitario: 0.99]

l1

Note que ao adicionarmos um objeto na lista a mesma vai se adaptando e criando novos índices abaixo dos já existentes, e assim como o *array* os índices de um *ArrayList* começam em 0.

Criando a classe Main

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner ler = new Scanner(System.in);

        ListaDeCompra l1 = new ListaDeCompra();
        int op;
        do{
            System.out.println("DIGITE");
            System.out.println("1- Cadastrar Produto");
            System.out.println("2- Visualizar Produtos");
            System.out.println("3- Ver quantidade de produtos");
            System.out.println("4- Apagar todos os produtos");
            System.out.println("5- Ver total de compras");
            System.out.println("0- Sair");
            op = ler.nextInt();
            switch(op){
                case 1:
                    Produto p1 = new Produto();
                    System.out.println("Digite o nome do produto:");
                    ler.nextLine();
                    p1.setNome(ler.nextLine());
                    do{
                        System.out.println("Digite a quantidade:");
                        p1.setQuantidade(ler.nextInt());
                        if(p1.getQuantidade()<=0){
                            System.out.println("Erro! Quantidade invalida.");
                        }
                    }while(p1.getQuantidade()<=0);
                    do{
                        System.out.println("Digite o valor unitario:");
                        p1.setValorUnitario(ler.nextDouble());
                        if(p1.getValorUnitario()<=0){
                            System.out.println("Erro! Valor invalido.");
                        }
                    }while(p1.getValorUnitario()<=0);
                    l1.adicionarProduto(p1);
                    System.out.println("Cadastrado com sucesso!");
                    break;
```

Figura 4

Perceba que na classe Main, temos no *case* 1 o cadastro de um produto, neste exemplo utilizamos um laço *do while* para forçar o usuário informar apenas valores corretos, exemplo quantidade só pode ser de 1 para cima e valores não podem jamais ser negativos ou nulos.

```
34              case 2:
35                  if(l1.getLista().isEmpty()){
36                      System.out.println("Não ha produtos");
37                  } else{
38                      System.out.println(l1);
39                  }
40                  break;
```

Figura 5

Observando a figura 5 podemos ver que foi utilizado uma condicional para ver se há objetos cadastrados, na linha 35 testamos se a lista está vazia, caso esteja retornará ao usuário: "Não há produtos." Se não chamará o toString da classe ListaDeCompra.

```
48              case 3:
49                  System.out.println("Total de cadastros: "
50                          +l1.obterQuantidadeDeProdutos());
51                  break;
52              case 4:
53                  if(l1.getLista().isEmpty()){
54                      System.out.println("Nao ha produtos para excluir.");
55                  } else{
56                      int confirma;
57                      System.out.println("Tem certeza que deseja EXCLUIR TUDO?\n1-SIM\n2-NAO");
58                      confirma = ler.nextInt();
59                      if(confirma==1){
60                          l1.apagarTudo();
61                          System.out.println("Produtos excluidos com sucesso.");
62                      }else{
63                          System.out.println("Opcao cancelada.");
64                      }
65                  }
66                  break;
67              case 5:
68                  System.out.println("Total da lista R$ "
69                          +l1.calcularTotalDaLista());
70                  break;
```

Figura 6

Repare que no *case* 3 apenas imprimimos a quantidade de produtos na tela, porém no *case* 4 – apagar todo os produtos, primeiro verificamos se a lista não está vazia, caso esteja retorna ao usuário a mensagem "Não há produtos para excluir." Se não, criamos uma variável confirma que receberá 1 confirmando a exclusão de todos os dados e qualquer outro número cancelando a operação. No caso de exclusão de todos os dados, uma confirmação é muito importante.

No *case* 5 apenas mostramos o total em valor de compras, portanto basta invocarmos o método calcularTotalDaLista() presente na classe ListaDeCompras.

```
71                  case 0:
72                      System.out.println("Sistema encerrado.");
73                      break;
74                  default:
75                      System.out.println("Opcao invalida.");
76              }
77          }while(op!=0);
78      }
79  }
```

Figura 7

BIBLIOGRAFIA

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