PROGRAMAÇÃO I - CCT0827

Semana Aula: 14

Unidade 4 - Coleções

Tema

Coleções

Palavras-chave

Coleção, ArrayList, Wrapper

Objetivos

O aluno deverá ser capaz de:

- Compreender o conceito e o objetivo do uso de coleções
- Compreender a classe ArrayList
- Realizar aplicações com ArrayList
- Compreender o uso de classes Wrapper

Estrutura de Conteúdo

Introdução

- A linguagem Java possui um conjunto de classes que servem para armazenar, na memória, coleções de objetos.
- Tais classes possuem a vantagem de não termos que saber, de antemão, a quantidade de elementos que iremos armazenar (que é uma grande desvantagem dos vetores).
- Todas as coleções estão definidas no pacote java.util.
- As coleções em Java são definidas a partir de 4 interfaces principais:
  - Collection
  - Set

- List: define métodos para manipulação de listas.
- Mapa

- Algumas classes que implementam estas interfaces são : ArrayList, LinkedList (implementam interface List), entre outras.

Classe ArrayList

O ArrayList é uma classe concreta que implementa a interface List, ou seja, uma lista de objetos. Cada objeto armazenado no ArrayList possui um índice e através desse índice, é possível recuperar determinado objeto da lista.

Note que a interface List é sub-interface de Collection.

A interface Collection define vários métodos básicos para manipulação de conjuntos e listas de objetos:

- boolean add(Object): adiciona um objeto à coleção. Retorna *true* ou *false* para indicar se a operação foi bem sucedida ou não.
- boolean remove(Object): remove o objeto especificado da coleção. Retorna *false* se o objeto não pertence à coleção.
- boolean contains(Object): procura por um determinado objeto na coleção e retorna *true* ou *false*, se o objeto existir ou não. A comparação é feita pelo método equals().
- int size(): retorna a quantidade de objeto presentes na coleção.
- boolean isEmpty(): retorna *true* se a coleção está vazia ou *false* caso contrário.
- void clear(): remove todos os objeto da coleção.

Além dos métodos definidos na interface Collection, a interface List acrescenta mais alguns métodos importantes:

- void add(índice, Object): adiciona o objeto à coleção na posição do índice.
- Object get(índice): recupera o objeto de determinada posição da lista (da mesma forma como é feito com vetores).
- Object set(índice, Object): substitui o objeto da posição do índice pelo novo objeto. Retorna o objeto que estava armazenado anteriormente.

· Object remove(índice): remove o objeto de determinada posição da lista. Retorna o objeto removido.

· int indexOf(Object): retorna o índice da primeira ocorrência de um objeto ou ?1 se ele não existir na lista. A comparação é feita pelo método equals().

· int lastIndexOf(Object): retorna o índice da última ocorrência de um objeto ou ?1 se ele não existir na lista. A comparação é feita pelo método equals().

É importante notar o efeito dos métodos na lista de objetos :

Método / Descrição :

· boolean add(Object) : Adiciona o objeto sempre no final da lista.

· void add(índice, Object) : Adiciona um objeto na i-ésima posição e move os objetos subsequentes para a posição posterior (índice deve ter valor de 0 a size()).

· boolean remove(Object) : Remove o objeto da lista e move os objetos subsequentes para a posição anterior.

· Object remove(índice) : Remove o objeto da i-ésima posição da lista e move os objetos subsequentes para a posição anterior (índice deve ter valor de 0 a size()-1).

· Object set(índice, Object) : Substitui o elemento da i-ésima posição pelo novo objeto (índice deve ter valor de 0 a size()-1).

Exemplo 1: Adicionando e removendo elementos da lista.

```
import java.util.ArrayList;

public class ExemploArrayList {

  public static void main(String[] args) {

     ArrayList lista = new ArrayList();

     lista.add("Dinardo");

     lista.add("Rosa");

     lista.add("Dinardo");

     lista.add("Rafael");
```

```
      lista.add("Luís");
      lista.remove("Carlos");
      lista.remove("Rafael");
      lista.remove("Dinardo");
  }
}
```

Exemplo 2: Podemos armazenar objetos de tipos diferentes.

```
import java.util.ArrayList;
public class ExemploArrayList {
   public static void main(String[] args) {

   // Definido dessa forma, o ArrayList armazena objetos do tipo Object.
      ArrayList lista = new ArrayList();

      lista.add("Dinardo");
      lista.add("Rosa");
      lista.add(10);
      lista.add(2465);
      lista.add(3.14159);
      lista.add('A');
  }
}
```

Considere lista do exemplo anterior, então podemos ter :

O método **get(i)** recupera o elemento da i-ésima posição. A primeira posição é ZERO.

```
System.out.println(lista.get(0));
System.out.println(lista.get(3));
System.out.println(lista.get(4));
System.out.println(lista.get(5));
```

O método **set(i, obj)** altera o elemento da i-ésima posição.

```
lista.set(1, 200);
lista.set(2, "Andre");
```

O método **remove(i)** remove o elemento da i-ésima posição.

```
lista.remove(2);
lista.remove(4);
```

Para percorrer um ArrayList podemos adotar uma das 3 estratégias:

- for
- for each
- Iterator ou ListIterator

Exemplo : usando um for tradicional.

```
import java.util.ArrayList;
public class ExemploArrayList {
   public static void main(String[] args) {
      ArrayList lista = new ArrayList();
      lista.add("Dinardo");
      lista.add("Rosa");
      lista.add(10);
      lista.add(2465);
      lista.add(3.14159);
      lista.add('A');

      for (int i = 0; i < lista.size(); i++)
         System.out.println(lista.get(i));
   }
}
```

Exemplo: usando um for each.

```
import java.util.ArrayList;
public class ExemploArrayList {
   public static void main(String[] args) {
      ArrayList lista = new ArrayList();
      lista.add("Dinardo");
      lista.add("Rosa");
      lista.add(10);
```

```
        lista.add(2465);
        lista.add(3.14159);
        lista.add('A');

        for (Object obj : lista)
            System.out.println(obj);
    }
}
```

Na interface Collection também é definido o método:

- Iterator iterator(): retorna um objeto Iterator que permite percorrer os objetos da coleção de forma sequencial.

A classe Iterator implementa os seguintes métodos:

- boolean hasNext(): informa se existe um próximo elemento na coleção.
- Object next(): retorna o próximo elemento da coleção.

Como é possível perceber, o Iterator só permite varrer a coleção em uma única direção: do início para o fim.

Na interface List também é definido o método:

- ListIterator listIterator(): retorna um objeto ListIterator que permite percorrer os objeto da lista da frente para trás ou vice-versa.

A classe ListIterator possui todos os métodos da Iterator, mais:

- boolean hasPrevious(): informa se existe um elemento anterior na lista.
- Object previous(): retorna o elemento anterior da lista.

Como é possível perceber, o ListIterator permite varrer a lista do início para o fim ou vice-versa.

Exemplo: usando um ListIterator.

```
import java.util.ArrayList;
public class ExemploArrayList {

  public static void main(String[] args) {
     ArrayList lista = new ArrayList();

      lista.add("Dinardo");
      lista.add("Rosa");
      lista.add(10);
      lista.add(2465);
      lista.add(3.14159);
      lista.add('A');

      ListIterator it = lista.listIterator();
      while (it.hasNext()) {
         Object obj = it.next();
         System.out.println(obj);
      }
  }
}
```

Nos exemplos anteriores armazenamos na lista, objetos já existentes na linguagem Java (Strings, caracteres, inteiros e reais). Entretanto, podemos armazenar na lista nossos próprios objetos. Por exemplo, uma lista de objetos de uma classe Aluno.

Exemplo :

```
public class Aluno {
  private int matricula;
  private String nome;
  public Aluno(int matricula, String nome) {
    this.matricula = matricula;
    this.nome = nome;
  }
  public String toString() {
    return String.format("Matricula: %d    Nome: %s\n", matricula, nome);
  }
}
import java.util.ArrayList;
public class ExemploArrayList {
   public static void main(String[] args) {
      ArrayList alunos = new ArrayList();
      alunos.add(new Aluno(123, "Joao"));
      alunos.add(new Aluno(456, "Rosa"));
      alunos.add(new Aluno(789, "Ana"));
      alunos.add(new Aluno(246, "Lucas"));

      for (int i = 0; i < alunos.size(); i++)
          System.out.println(alunos.get(i));
  }
}
```

Importante:

Para usar os métodos:

- boolean remove(Object)
- boolean contains(Object)
- int indexOf(Object)
- int lastIndexOf(Object)

as classes armazenadas no ArrayList têm que, obrigatoriamente, implementar o método

boolean equals(Object)

Caso esse método não seja implementado, os métodos acima não funcionarão corretamente. Isso acontece porque os métodos acima usam o método equals para procurar o objeto desejado.

Classe Wrapper

- Em Java os tipos básicos int, long, byte, short, float, double, char e boolean não são objetos.
- Entretanto, a maioria das classes da biblioteca Java (como as classes que implementam as coleções, por exemplo), trabalham somente com objetos.
- Para resolver esse problema foram definidas classes *wrapper*, ou seja, classes que encapsulam os tipos básicos para que estes possam ser tratados como objetos.

Classe Wrapper	**/ tipo armazenado na classe**
Integer	int
Long	long
Byte	byte
Short	short
Float	float
Double	double

Character	char
Boolean	boolean

Na maior parte das vezes, o Java converte o tipo básico em uma classe wrapper (*boxing*) ou uma classe wrapper em um tipo básico (*unboxing*).

Exemplo :

```
int x = 10;

Integer a = 10;

Integer b = x + 1;

int y = a;

a = a + 1;

b++;
```

Nas classes *wrapper* estão definidos métodos para converter tipos básicos em classes (*boxing*) e classes em tipos básicos (*unboxing*).

Classe Wrapper	**Tipo básico à Classe (*boxing*)**	**Classe à tipo básico (*unboxing*)**
Integer	Integer n = Integer.valueOf(10)	int x = n.intValue()
Float	Float n = Float.valueOf(10.5f)	float x = n.floatValue()
Double	Double n = Double.valueOf(10.5)	double x = n.doubleValue()

Long	Long n = Long.valueOf(10)	long x = n.longValue()
Short	Short n = Short.valueOf(10)	short x = n.shortValue()
Byte	Byte n = Byte.valueOf(10)	byte x = n.byteValue()
Character	Character c = Character.valueOf('A')	char ch = c.charValue()
Boolean	Boolean b = Boolean.valueOf(true)	boolean bl = b.booleanValue()

Quando armazenamos inteiros, reais ou caracteres em uma lista, o Java faz automaticamente o *boxing*, ou seja, ele cria objetos usando as classes wrapper e insere esses objetos na lista.

Exemplo : Seja o trecho abaixo. Note que para armazenar esses valores são usadas as classes Integer, Double e Character.

```
ArrayList lista = new ArrayList();

lista.add(10);

lista.add(2465);

lista.add(3.14159);

lista.add('A');

lista.add('C');
```

Quando temos na lista objetos do tipo Integer, Double, Character, etc., temos que usar o *casting* para obter o valor de dentro do objeto.

Exemplo : Seja o trecho abaixo.

```
ArrayList lista = new ArrayList();
```

```
lista.add(10);
lista.add(2465);
lista.add(3.14159);
lista.add('A');
lista.add('C');
int n1    = (int) lista.get(0);
int n2    = (int) lista.get(1);
double n3 = (double) lista.get(2);
char c1   = (char) lista.get(3);
char c2   = (char) lista.get(4);
```

As classes *wrapper* possuem, ainda, métodos estáticos úteis para converter strings em tipos específicos. Caso a conversão não seja possível, será disparada uma exceção.

Classe Wrapper	Conversão de String
Integer	int Integer.parseInt(String s)
Float	float Float.parseFloat(String s)
Double	double Double.parseDouble(String s)
Long	long Long.parseLong(String s)
Short	short Short.parseShort(String s)

Byte	byte Byte.parseByte(String s)

Estratégias de Aprendizagem

Para que o aprendizado seja proveitoso, o aluno deve se envolver ativamente na participação da aula, deve realizar as tarefas propostas, realizar testes por conta própria nos programas desenvolvidos e compartilhar sua experiência/conclusão com todos.

Toda tarefa realizada pode ser conferida com o professor, para que haja certeza se está ou não correta.

Indicação de Leitura Específica

Aplicação: articulação teoria e prática

Exercícios :

1) Faça um programa em Java que leia strings e armazene-as em uma lista até que o usuário digite uma string vazia. Não permita que sejam armazenadas strings duplicadas na lista. Ao final, imprima a lista na ordem inversa.

2) Faça um programa em Java que leia uma string e imprima quantas vezes as letras de A a Z aparecem na string. Dica: Use uma lista para armazenar as letras lidas e outra lista para contar quantas vezes as letras aparecem.

GABARITO :

1) Veja com seu professor, por favor.

2)

```
package ex46;

import java.util.ArrayList;
```

```
import java.util.Scanner;

public class Ex46 {

   public static void main(String[] args) {
      Scanner t = new Scanner(System.in);
      ArrayList letras = new ArrayList();
      ArrayList contador = new ArrayList();
      String str;

      // Le a string
      System.out.println("Digite uma string:");
      str = t.nextLine();

      // Retira os espacos do fim e do inicio
      str = str.trim();

      // Converte tudo para maiuscula
      str = str.toUpperCase();

      // Percorre a string caracter por caracter
      for (int i = 0; i < str.length(); i++) {
         // Pega o caracter da i-esima posicao
         char c = str.charAt(i);

         // Verifica se caracter esta no intervalo de A a Z
         if (c >= 'A' && c <= 'Z') {
            // Verificar se a letra ja existe na lista de letras
            if (letras.contains(c)) {
               // Incrementa o contador da letra
               // Recuperar a posicao da letra na lista
               int p = letras.indexOf(c);

               // Incrementar o contador na posicao P
               int cont = (int) contador.get(p);
               cont++;
               contador.set(p, cont);
            }
            else {
               // Inserir a letra na lista de letras com contador = 1
               letras.add(c);

               // Inserir o 1 no contador
               contador.add(1);
            }
         }
```

```
        }

        System.out.println("Estatistica:");
        for (int i = 0; i < letras.size(); i++)
            System.out.printf("%c   %d\n", letras.get(i), contador.get(i));
    }
}
```

Considerações Adicionais