Estrutura de Dados Orientada a Objetos

Prof. Evandro César Freiberger

Instituto Federal de Mato Grosso
Departamento da Área de Informática
evandro.freiberger@cba.ifmt.edu.br

2017

Sumário

Classe Utilitária Arrays

Classe utilitária com diversos métodos estáticos para manipulação de vetores.

- Principais métodos:
  - void Arrays.sort(vetor) - ordena os elementos de um vetor
  - boolean Arrays.equals(vetor1, vetor2) - compara dois vetores
  - int Arrays.binarySearch(vetor, chave) - realiza uma busca binária em um vetor
  - void Arrays.fill(vetor, valor) - preenche um vetor com um determinado valor
  - String toString(vetor) - devolve uma String com os elementos formatados "[e1, e2, ..., eN]"

Classe Utilitária Arrays (1)

Exemplo do uso dos métodos *sort* e *toString* da classe Arrays

```
1  package apps;
2  import java.util.Arrays;
3  public class ExemploArrays01 {
4
5      public static void main(String[] args) {
6
7          int vet[] = {4, 6, 8, 2, 1, 9, 0, 7, 5, 3};
8
9          Arrays.sort(vet);
10
11         System.out.println(Arrays.toString(vet));
12     }
13 }
```

Classe Utilitária Arrays (1)

Exemplo do uso do método *equals* da classe Arrays

```
1  package apps;
2  import java.util.Arrays;
3  public class ExemploArrays02 {
4
5      public static void main(String[] args) {
6
7          int vet1[] = {4, 6, 8, 2, 1, 9, 0, 7, 5, 3};
8          int vet2[] = {4, 6, 8, 2, 1, 9, 0, 7, 5, 3};
9
10         System.out.println("Vet1 => "+Arrays.toString(vet1));
11         System.out.println("Vet2 => "+Arrays.toString(vet2));
12
13         if(Arrays.equals(vet1, vet2)){
14             System.out.println("Sao iguais");
15         }else{
16             System.out.println("Sao diferentes");
17         }
18
19         Arrays.sort(vet1);
20
21         System.out.println("Vet1 => "+Arrays.toString(vet1));
22         System.out.println("Vet2 => "+Arrays.toString(vet2));
23
24         if(Arrays.equals(vet1, vet2)){
25             System.out.println("Sao iguais");
26         }else{
27             System.out.println("Sao diferentes");
28         }
```

Classe Utilitária Arrays (2)

```
29
30          Arrays.sort(vet2);
31
32          System.out.println("Vet1 => "+Arrays.toString(vet1));
33          System.out.println("Vet2 => "+Arrays.toString(vet2));
34
35          if(Arrays.equals(vet1, vet2)){
36              System.out.println("Sao iguais");
37          }else{
38              System.out.println("Sao diferentes");
39          }
40
41      }
42  }
```

Classe Utilitária Arrays (1)

Exemplo do uso do método *fill* da classe Arrays. O método *fill* preenche o vetor com um valor passado.
Opcionalmente pode ser usado parâmetros para definir uma faixa de preenchimento (fromIndex e toIndex). O valor fromIndex define o início do preenchimento (inclusive) e o valor toIndex define até onde vai o prenchimento (exclusive).

```
1  package apps;
2  import java.util.Arrays;
3  public class ExemploArrays03 {
4
5      public static void main(String[] args) {
6
7          int vet1[] = new int[10];
8          float vet2[] = new float[10];
9          char vet3[] = new char[10];
10
11         Arrays.fill(vet1, 1);
12         Arrays.fill(vet2, 2.5f);
13         Arrays.fill(vet3, 0, 5, 'x');
14         Arrays.fill(vet3, 5, 10, 'y');
15
16         System.out.println(Arrays.toString(vet1));
17         System.out.println(Arrays.toString(vet2));
18         System.out.println(Arrays.toString(vet3));
```

Classe Utilitária Arrays (2)

```
19         }
20 }
```

Classe Utilitária Arrays (1)

Exemplo do método de busca *binarySearch* da classe Arrays. O vetor deve estar previamente ordenado para que a busca seja realizada.

```
1  package apps;
2  import java.util.Arrays;
3  public class ExemploArrays04 {
4
5      public static void main(String[] args) {
6          int resu;
7          int vet[] = {4, 6, 8, 2, 1, 9, 0, 7, 5, 3};
8
9          resu = Arrays.binarySearch(vet, 2);
10         System.out.println("Resultado = "+resu);
11
12         Arrays.sort(vet);
13
14         resu = Arrays.binarySearch(vet, 2);
15         System.out.println("Resultado = "+resu);
16
17         vet[3] = 2;
18         System.out.println("Vet = "+Arrays.toString(vet));
19
20         resu = Arrays.binarySearch(vet, 2);
21         System.out.println("Resultado = "+resu);
22
23     }
24 }
```

Classe Utilitária Arrays (1)

Ordenação em ordem decrescente com método sort. Ordem decrescente não é possível com tipos primitivos, visto que é necessário um comparador específico da API Collections do Java.

```
1  package apps;
2  import java.util.Arrays;
3  import java.util.Collections;
4  public class ExemploArrays05 {
5
6      public static void main(String[] args) {
7
8          Integer vet1[] = {4, 6, 8, 2, 1, 9, 0, 7, 5, 3};
9
10         System.out.println("Vet1 => "+Arrays.toString(vet1));
11
12         Arrays.sort(vet1);
13
14         System.out.println("Vet1 => "+Arrays.toString(vet1));
15
16         Arrays.sort(vet1, Collections.reverseOrder());
17
18         System.out.println("Vet1 => "+Arrays.toString(vet1));
19
20     }
21 }
```

Coleções em Java

Uma coleção é uma estrutura de dados que permite armazenar vários objetos.

As coleções em si são classes que oferecem serviços para a realização operações sobre os objetos que armazenam, os serviços mais comuns são:

- Adição de elementos
- Remoção de elementos
- Acesso aos elementos
- Pesquisa de elementos
- Obter informações da coleção
- Algum “tipo” de coerção (como uma “conversão” de uma coleção para outra)

Tipos de Coleções em Java

Em Java o termo coleção é utilizado para representar as estruturas de dados dinâmicas. Uma estrutura de dados dinâmica pode sofrer

mudanças em sua coleção de elementos durante a execução do programa.

Existem três grupos de coleções, são eles:

- Lista
- Conjunto
- Mapa

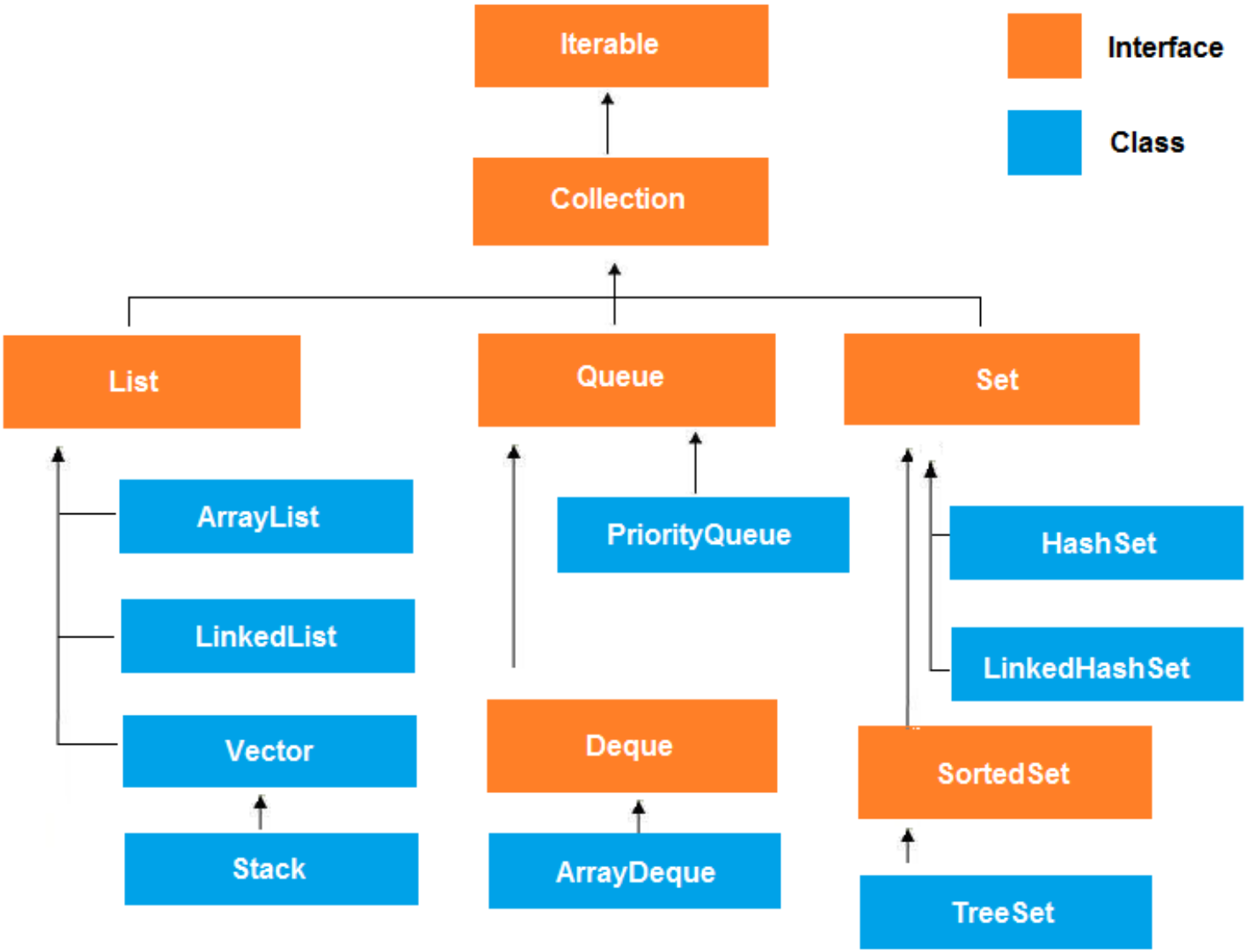
Tipos de Coleções em Java

A escolha de um tipo de coleção é feita em função da própria natureza dos dados e como deseja-se recuperar esses dados.

- Listas – estruturas lineares (vetores dinâmicos) que armazenam uma sequencia de elementos, onde a ordem e a contagem deles são importantes.
- Conjunto – implementam o conceito de conjuntos, que implica, por exemplo em não armazenar redundância de elementos, a ordem não é importante. Podem ser aplicadas as operações: união, interseção e diferença.
- Mapas – implementam o conceito de dicionários, onde os elementos são armazenados com pares chave-valor.

Coleções em Java

Em Java as implementações das coleções seguem a seguinte hierarquia:



Coleções em Java (1)

Collection é uma interface que define um conjunto genérico de métodos comuns as coleções que são implementadas em Java; Vamos analisar esses métodos:

- add(Object o) => adiciona um elemento/objeto na coleção e retorna true se a operação foi bem sucedida.
- addAll(Collection c) => adiciona todos os elementos/objetos da coleção e retorna true em caso de sucesso.
- clear() => remove todos os elementos da coleção.
- contains(Object o) => retorna true se a coleção contém o elemento/objeto.
- containsAll(Collection c) => retorna true se todos os elementos de c então presentes na coleção.
- isEmpty() => retorna true se a coleção estiver vazia.
- remove(Object o) => remove o objeto da coleção.

Coleções em Java (2)

- removeAll(Collection c) => remove todos os objetos presentes em c da coleção em questão.
- retainALL(Collection c) => remove da coleção todos os elementos que não estão presentes em c.
- size() => retorna o número de elementos presentes na coleção.
- toArray() => retorna um vetor de objetos (Object[ ]).
- interator() => retorna uma coleção navegável do tipo Iterator.

Implementações de Listas em Java

É uma coleção de elementos organizados de forma linear.

Cada elemento tem um antecessor (exceto o primeiro) e um sucessor (exceto o último).

Normalmente implementada como um vetor, representada pelas classes *Vector* e *ArrayList*, ou lista encadeada, representada pela classe *LinkedList*.

Todas as implementações de listas em Java implementam a interface *List*.

List por sua vez é subtipo de *Collection*.

Implementações de Listas em Java

As operações mais importantes das coleções do tipo Lista são:

- Adição – adicionar um objeto em qualquer lugar da lista (início, meio, fim), fornecendo o índice desejado.
- Remoção – remover um objeto presente em qualquer lugar da lista, fornecendo o índice desejado.
- Acesso – recuperar o elemento de qualquer posição da lista, fornecendo o índice desejado.
- Pesquisa – descobrir se certo elemento está na lista ou o índice do mesmo.
- Obter informações da lista – obter o número de elementos da coleção.

Interface List (1)

A interface *List* adiciona os seguintes métodos, dos já conhecidos da interface *Collection*:

- add (int i, Object o) => adiciona um elemento/objeto na posição especificada pelo índice i, retorna true se a operação foi bem sucedida.
- addALL (int i, Collection c) => adiciona os elementos/objetos da coleção c a partir da posição especificada pelo índice i, retorna true se a operação foi bem sucedida.
- get(int i) => retorna o elemento/objeto indicado pela posição i.
- indexOf(Object o) => retorna a posição da primeira ocorrência do elemento/objeto na lista. Se não for encontrado, retorna -1.
- lastIndexOf(Object o) => retorna a posição da ultima ocorrência do elemento/objeto na lista. Se não encontrar, retorna -1.
- listInterator() => retorna um objeto ListInterator para navegação nos dois sentidos da lista.

Interface List (2)

- listInterator(int i) => retorna um objeto ListInterator para navegação a partir da posição i.
- remove(int i) => remove o elemento/objeto da posição i. Retorna o objeto removido.
- set(int i, Object o) => substitui o elemento da posição i pelo elemento/objeto o. Retorna o objeto sobreposto.
- subList(int ini, int fim) => retorna uma sublista delimitada por uma posição inicial(inclusive) e final(exclusive).

Implementações de Listas em Java

É uma coleção de elementos organizados de forma linear.

Cada elemento tem um antecessor (exceto o primeiro) e um sucessor (exceto o último).

Normalmente implementada como um vetor, representada pelas classes *Vector* e *ArrayList*, ou lista encadeada, representada pela classe *LinkedList*.

Todas as implementações de listas em Java implementam a interface *List*.

List por sua vez é subtipo de *Collection*.

Implementações de Listas em Java

As operações mais importantes das coleções do tipo Lista são:

- Adição – adicionar um objeto em qualquer lugar da lista (início, meio, fim), fornecendo o índice desejado.
- Remoção – remover um objeto presente em qualquer lugar da lista, fornecendo o índice desejado.
- Acesso – recuperar o elemento de qualquer posição da lista, fornecendo o índice desejado.
- Pesquisa – descobrir se certo elemento está na lista ou o índice do mesmo.
- Obter informações da lista – obter o número de elementos da coleção.

Interface List (1)

A interface *List* adiciona os seguintes métodos, dos já conhecidos da interface *Collection*:

- add (int i, Object o) => adiciona um elemento/objeto na posição especificada pelo índice i, retorna true se a operação foi bem sucedida.
- addALL (int i, Collection c) => adiciona os elementos/objetos da coleção c a partir da posição especificada pelo índice i, retorna true se a operação foi bem sucedida.
- get(int i) => retorna o elemento/objeto indicado pela posição i.
- indexOf(Object o) => retorna a posição da primeira ocorrência do elemento/objeto na lista. Se não for encontrado, retorna -1.
- lastIndexOf(Object o) => retorna a posição da ultima ocorrência do elemento/objeto na lista. Se não encontrar, retorna -1.
- listInterator() => retorna um objeto ListInterator para navegação nos dois sentidos da lista.

Interface List (2)

- listInterator(int i) => retorna um objeto ListInterator para navegação a partir da posição i.
- remove(int i) => remove o elemento/objeto da posição i. Retorna o objeto removido.
- set(int i, Object o) => substitui o elemento da posição i pelo elemento/objeto o. Retorna o objeto sobreposto.
- subList(int ini, int fim) => retorna uma sublista delimitada por uma posição inicial(inclusive) e final(exclusive).

Classe Vector

A classe *Vector* implementa a interface *List* com as seguintes características:

- Implementa uma estrutura de dados do tipo vetor.
- É redimensionável, ou seja, muda de tamanho, mas através de cópias para um novo vetor (resolvido internamente).
- Não é uma lista encadeada que faz alocação dinâmica de memória.
- Inicia com alocação padrão com a capacidade de 10 elementos.
- Quando o vetor é preenchido, a classe Vector dobra o tamanho do vetor.

Classe Vector

Os métodos dessa estrutura são todos aquelas provenientes da interface List, com os seguintes acréscimos:

- copyInto(Object[ ] array) => faz uma cópia dos elemento para um vetor passado como argumento.
- trimToSize( ) => reduz o tamanho do vetor para o número efetivo de elementos.
- ensureCapacity(int min) => aumenta a capacidade do vetor para no mínimo o valor do argumento passado.
- setSize(int num) => define o tamanho do vetor com base no argumento passado.
- capacity( ) => retorna o tamanho do vetor.

Exemplo01 - Classe Vector

Exemplo de instanciação e métodos: size(), capacity(), add() e clear().

```
1  package apps;
2  import java.util.Vector;
3  public class ExemploVector01 {
4      public static void main(String[] args) {
5
6          Vector vetor = new Vector();
7          System.out.println("Tamanho = "+vetor.size());
8          System.out.println("Capacidade = "+vetor.capacity());
9          vetor.add(2);
10         vetor.add(2);
11         vetor.add(2.5f);
12         vetor.add("Teste Vector");
13         System.out.println("Tamanho = "+vetor.size());
14         System.out.println("Capacidade = "+vetor.capacity());
15         vetor.clear();
16         System.out.println("Tamanho = "+vetor.size());
17         System.out.println("Capacidade = "+vetor.capacity());
18         for(int x=1; x <= 15; x++){
19             vetor.add(x);
20         }
21         System.out.println("Tamanho = "+vetor.size());
22         System.out.println("Capacidade = "+vetor.capacity());
23     }
24 }
```

Exemplo02 - Classe Vector (1)

Exemplo de instanciação com delimitador de capacidade e incremento, e os métodos: get(), contains(), add() e clear().

```
1  package apps;
2  import java.util.Vector;
3  public class ExemploVector02 {
4      public static void main(String[] args) {
5          Vector<Integer> vetor = new Vector(5, 3);
6          System.out.println("Tamanho = "+vetor.size());
7          System.out.println("Capacidade = "+vetor.capacity());
8          for(int x=1; x <= 6; x++){
9              vetor.add(x*2);
10         }
11         System.out.println("Tamanho = "+vetor.size());
12         System.out.println("Capacidade = "+vetor.capacity());
13         for(Integer valor : vetor){
14             System.out.println("Valor = "+valor);
15         }
16         System.out.println("Valor aleatorio = "+vetor.get(2));
17         if (vetor.contains(10)){
18             System.out.println("O elemento 10 esta presente");
19         }else{
20             System.out.println("O elemento 10 nao esta presente");
21         }
22         if (vetor.contains(20)){
23             System.out.println("O elemento 20 esta presente");
24         }else{
25             System.out.println("O elemento 20 nao esta presente");
26         }
```

Exemplo02 - Classe Vector (2)

```
27         }
28 }
```

Exemplo03 - Classe Vector (1)

Exemplo dos métodos: toString(), indexOf() e remove.

```
1  package apps;
2  import java.util.Vector;
3  public class ExemploVector03 {
4      public static void main(String[] args) {
5          Vector<Integer> vetor = new Vector(5, 3);
6          for(int x=1; x <= 6; x++){
7              vetor.add(x*2);
8          }
9          System.out.println(vetor.toString());
10         System.out.println("Tamanho = "+vetor.size());
11         System.out.println("Capacidade = "+vetor.capacity());
12         System.out.println("Posicao do valor 8 = "+vetor.indexOf(8));
13         vetor.trimToSize();
14         System.out.println("Tamanho = "+vetor.size());
15         System.out.println("Capacidade = "+vetor.capacity());
16         //vetor.remove(10);
17         vetor.remove(new Integer(10));
18         System.out.println(vetor.toString());
19         System.out.println("Tamanho = "+vetor.size());
20         System.out.println("Capacidade = "+vetor.capacity());
21     }
22 }
```

Classe ArrayList

A implementação da estrutura linear *ArrayList* implementa a interface *List*, com as seguintes características:

- Também é uma estrutura de dados baseada em vetor.
- Redimensionável, ou seja, também não é uma lista encadeada, ela é similar a classe *Vector*.
- Inicia com uma alocação padrão e aumenta em 50% toda vez que o vetor é preenchido.
- Os métodos dessa estrutura são todos aqueles provenientes de List.
- A principal diferença entre *Vector* e *ArrayList* é que Vector é preparada para trabalhar em sincronismo, já *ArrayList* não.
- Caso não seja necessário o sincronismo, *ArrayList* tem melhor performance.

Exemplo01 - Classe ArrayList (1)

```
1  package apps;
2  import java.util.ArrayList;
3  public class ExemploArrayList01 {
4      public static void main(String[] args) {
5
6          ArrayList lista = new ArrayList();
7          System.out.println("Vazia = "+lista.isEmpty());
8          System.out.println("Tamanho = "+lista.size());
9          lista.add(2);
10         lista.add(2);
11         lista.add(2.5f);
12         lista.add("Teste ArrayList");
13         System.out.println("Vazia = "+lista.isEmpty());
14         System.out.println("Tamanho = "+lista.size());
15         lista.clear();
16         System.out.println("Tamanho = "+lista.size());
17         for(int x=1; x <= 7; x++){
18             lista.add(x*2);
19         }
20         System.out.println("Tamanho = "+lista.size());
21         for(int x = 0; x < lista.size(); x++){
22             System.out.println("lista["+x+"] = "+lista.get(x));
23         }
24     }
25 }
```

Exemplo02 - Classe ArrayList (1)

```
1  package apps;
2  import java.util.ArrayList;
3  import java.util.List;
4  public class ExemploArrayList02 {
5      public static void main(String[] args) {
6
7          ArrayList lista = new ArrayList();
8
9          lista.add("Estrutura");
10         lista.add("de");
11         lista.add("Dados");
12         lista.add("Orientada");
13         lista.add("a");
14         lista.add("Objetos");
15
16         System.out.println("Tamanho = "+lista.size());
17         System.out.println("toString() => "+lista.toString());
18         for(int x = 0; x < lista.size(); x++){
19             System.out.println("lista["+x+"] = "+lista.get(x));
20         }
21         System.out.println("Posicao de = "+lista.indexOf("Dados"));
22         System.out.println("Posicao de = "+lista.indexOf("dados"));
23         List lista2 = lista.subList(0, 3);
24         System.out.println("lista2.toString() => "+lista2.toString());
25     }
26 }
```

Exemplo03 - Classe Aluno1 (1)

```
1  package conceito;
2
3  public class Aluno1 {
4
5      private int matricula;
6      private String nome;
7      private int semestre;
8
9      public Aluno1(int matricula, String nome, int semestre) {
10         this.matricula = matricula;
11         this.nome = nome;
12         this.semestre = semestre;
13     }
14
15     public int getMatricula() {
16         return matricula;
17     }
18
19     public void setMatricula(int matricula) {
20         this.matricula = matricula;
21     }
22
23     public String getNome() {
24         return nome;
25     }
26
27     public void setNome(String nome) {
28         this.nome = nome;
29     }
30
```

Exemplo03 - Classe Aluno1 (2)

```
31      public int getSemestre() {
32          return semestre;
33      }
34
35      public void setSemestre(int semestre) {
36          this.semestre = semestre;
37      }
38  }
```

Exemplo03 - Classe ArrayList (1)

```
1  package apps;
2  import conceito.Aluno1;
3  import java.util.ArrayList;
4  public class ExemploArrayList03 {
5      public static void main(String[] args) {
6
7          ArrayList<Aluno1> lista = new ArrayList();
8
9          lista.add(new Aluno1(111, "Maria", 3));
10         lista.add(new Aluno1(222, "Antonio", 4));
11         lista.add(new Aluno1(333, "Joao", 1));
12         lista.add(new Aluno1(444, "Jose", 2));
13
14         System.out.println("Tamanho = "+lista.size());
15         System.out.println("toString() => "+lista.toString());
16         for(Aluno1 aluno : lista){
17             System.out.println(aluno);
18         }
19         Aluno1 alunoTemp = new Aluno1(333, "Joao", 1);
20         System.out.println("Posicao de = "+lista.indexOf(alunoTemp));
21         alunoTemp = lista.get(2);
22         System.out.println("Posicao de = "+lista.indexOf(alunoTemp));
23     }
24 }
```

Exemplo04 - Classe Aluno2 (1)

```
1  package conceito;
2
3  public class Aluno2 {
4
5      private int matricula;
6      private String nome;
7      private int semestre;
8
9      public Aluno2(int matricula, String nome, int semestre) {
10         this.matricula = matricula;
11         this.nome = nome;
12         this.semestre = semestre;
13     }
14
15     public int getMatricula() {
16         return matricula;
17     }
18
19     public void setMatricula(int matricula) {
20         this.matricula = matricula;
21     }
22
23     public String getNome() {
24         return nome;
25     }
26
27     public void setNome(String nome) {
28         this.nome = nome;
29     }
30
```

Exemplo04 - Classe Aluno2 (2)

```
31      public int getSemestre() {
32          return semestre;
33      }
34
35      public void setSemestre(int semestre) {
36          this.semestre = semestre;
37      }
38
39      @Override
40      public boolean equals(Object obj) {
41          if (obj == null) {
42              return false;
43          }
44          if (getClass() != obj.getClass()) {
45              return false;
46          }
47          final Aluno2 other = (Aluno2) obj;
48          if (this.matricula != other.matricula) {
49              return false;
50          }
51          return true;
52      }
53
54      @Override
55      public int hashCode() {
56          int hash = 7;
57          hash = 97 * hash + this.matricula;
58          return hash;
59      }
60
61      @Override
```

Exemplo04 - Classe Aluno2 (3)

```
62      public String toString() {
63          return "Aluno2{" + "matricula=" + matricula + ", nome=" + nome + ", semestre=" + semestre +
                '}';
64      }
65  }
```

Exemplo04 - Classe ArrayList (1)

```
1  package apps;
2  import conceito.Aluno2;
3  import java.util.ArrayList;
4  public class ExemploArrayList04 {
5      public static void main(String[] args) {
6
7          ArrayList<Aluno2> lista = new ArrayList();
8
9          lista.add(new Aluno2(111, "Maria", 3));
10         lista.add(new Aluno2(222, "Antonio", 4));
11         lista.add(new Aluno2(333, "Joao", 1));
12         lista.add(new Aluno2(444, "Jose", 2));
13
14         System.out.println("Tamanho = "+lista.size());
15         System.out.println("toString() => "+lista.toString());
16         for(Aluno2 aluno : lista){
17             System.out.println(aluno);
18         }
19         Aluno2 alunoTemp = new Aluno2(333, "Joao", 1);
20         System.out.println("Posicao de = "+lista.indexOf(alunoTemp));
21         alunoTemp = lista.get(2);
22         System.out.println("Posicao de = "+lista.indexOf(alunoTemp));
23     }
24 }
```

Exemplo05 - Classe Aluno3 (1)

```
1  package conceito;
2
3  public class Aluno3 implements Comparable<Aluno3>{
4
5      private int matricula;
6      private String nome;
7      private int semestre;
8
9      public Aluno3(int matricula, String nome, int semestre) {
10         this.matricula = matricula;
11         this.nome = nome;
12         this.semestre = semestre;
13     }
14
15     public int getMatricula() {
16         return matricula;
17     }
18
19     public void setMatricula(int matricula) {
20         this.matricula = matricula;
21     }
22
23     public String getNome() {
24         return nome;
25     }
26
27     public void setNome(String nome) {
28         this.nome = nome;
29     }
30
```

Exemplo05 - Classe Aluno3 (2)

```
31      public int getSemestre() {
32          return semestre;
33      }
34
35      public void setSemestre(int semestre) {
36          this.semestre = semestre;
37      }
38
39      @Override
40      public boolean equals(Object obj) {
41          if (obj == null) {
42              return false;
43          }
44          if (getClass() != obj.getClass()) {
45              return false;
46          }
47          final Aluno3 other = (Aluno3) obj;
48          if (this.matricula != other.matricula) {
49              return false;
50          }
51          return true;
52      }
53
54      @Override
55      public int hashCode() {
56          int hash = 7;
57          hash = 97 * hash + this.matricula;
58          return hash;
59      }
60
61      @Override
```

Exemplo05 - Classe Aluno3 (3)

```
62      public String toString() {
63          return "matricula=" + matricula + ", nome=" + nome;
64      }
65
66      @Override
67      public int compareTo(Aluno3 outro) {
68          return this.getNome().compareTo(outro.getNome());
69      }
70  }
```

Exemplo05 - Classe ArrayList (1)

```
1  package apps;
2  import conceito.Aluno3;
3  import java.util.ArrayList;
4  import java.util.Collections;
5  public class ExemploArrayList05 {
6      public static void main(String[] args) {
7
8          ArrayList<Aluno3> lista = new ArrayList();
9
10         lista.add(new Aluno3(111, "Maria", 3));
11         lista.add(new Aluno3(222, "Antonio", 4));
12         lista.add(new Aluno3(333, "Joao", 1));
13         lista.add(new Aluno3(444, "Jose", 2));
14
15         System.out.println("toString() => "+lista.toString());
16
17         Collections.sort(lista);
18         System.out.println("toString() => "+lista.toString());
19     }
20 }
```

Exemplo06 - Classe AlunoComparaNome (1)

```
1  package conceito;
2
3  import java.util.Comparator;
4
5  public class AlunoComparaNome implements Comparator<Aluno3> {
6
7      @Override
8      public int compare(Aluno3 o1, Aluno3 o2) {
9          return o1.getNome().compareTo(o2.getNome());
10     }
11
12 }
```

Exemplo06 - Classe AlunoComparaSemestre (1)

```
1  package conceito;
2
3  import java.util.Comparator;
4
5  public class AlunoComparaSemestre implements Comparator<Aluno3> {
6
7      @Override
8      public int compare(Aluno3 o1, Aluno3 o2) {
9          if(o1.getSemestre() < o2.getSemestre()){
10             return -1;
11         }else{
12             if(o1.getSemestre() > o2.getSemestre()){
13                 return 1;
14             }else{
15                 return 0;
16             }
17         }
18     }
19 }
```

Exemplo06 - Classe ArrayList (1)

```
1  package apps;
2  import conceito.Aluno3;
3  import conceito.AlunoComparaSemestre;
4  import conceito.AlunoComparaNome;
5  import java.util.ArrayList;
6  import java.util.Collections;
7  public class ExemploArrayList06 {
8      public static void main(String[] args) {
9
10         ArrayList<Aluno3> lista = new ArrayList();
11
12         lista.add(new Aluno3(111, "Maria", 3));
13         lista.add(new Aluno3(222, "Antonio", 4));
14         lista.add(new Aluno3(333, "Joao", 1));
15         lista.add(new Aluno3(444, "Jose", 2));
16
17         System.out.println("toString() => "+lista.toString());
18
19         Collections.sort(lista, new AlunoComparaNome());
20         System.out.println("toString() => "+lista.toString());
21
22         Collections.sort(lista, new AlunoComparaSemestre());
23         System.out.println("toString() => "+lista.toString());
24     }
25 }
```