ânima EDUCAÇÃO

UNISUL
UNIVERSIDADE DO SUL DE SANTA CATARINA

PROGRAMAÇÃO DE SOLUÇÕES COMPUTACIONAIS

Prof. Ricardo Ribeiro Assink

POO - *COLLECTIONS*

As ***Collections*** são estruturas de dados pré-definidas, fornecidas pela API do Java, que são empregadas para armazenar grupos de objetos.

Com estas classes é possível organizar, armazenar e ler dados sem a necessidade de saber exatamente como estes foram construídos (***encapsulamento***).

POO - *COLLECTIONS*

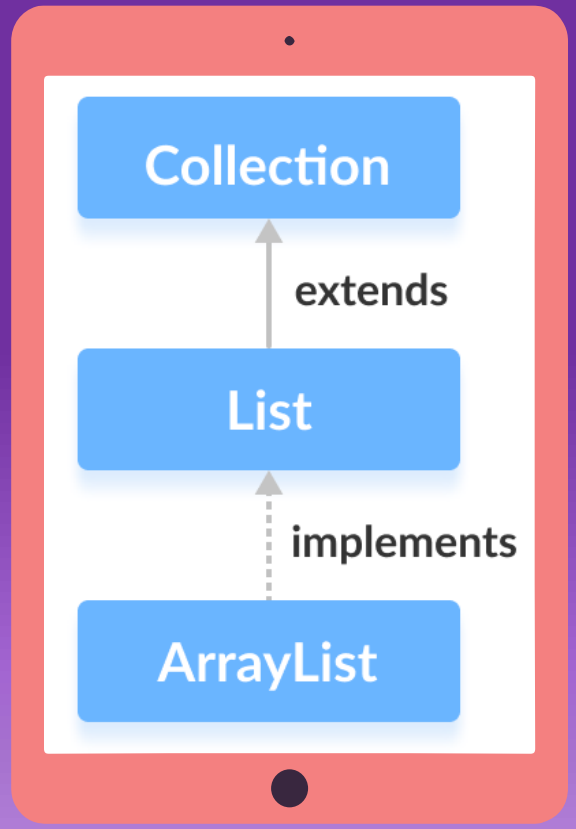
O framework ***Collections*** em Java fornece diversas interfaces e classes para manipulação de grupos de objetos, como por exemplo:

Interfaces

- List
- Set
- Queue
- Deque
- SortedSet

Classes

- *ArrayList*
- LinkedList
- Vector
- HashSet
- PriorityQueue
- LinkedHashSet
- TreeSet



Descrição de imagem:
Mostra a classe ArrayList implementando a classe List (interface) e a classe List estendendo a classe Collection

POO – *ArrayList*

Um array convencional (lembra de Vetores e Matrizes?) não tem seu tamanho modificado automaticamente durante a execução do programa.

Exemplo:

```
String vetor [] = new String[10];
```

A classe de coleção (collections class) ***ArrayList<>,*** pertencente ao pacote **java.util**, e que implementa a interface ***List***, resolve o problema de modificação do tamanho de um array em tempo de execução, pois ela pode mudar seu tamanho de forma dinâmica – ou seja, permite criar um array dinâmico.

Exemplo:

```
ArrayList<Aluno> listasimples = new ArrayList<>();
```

POO – *ArrayList*

Somente tipos não-primitivos podem ser usados para declarar variáveis e criar objetos em classes genéricas – e o *ArrayList* é uma classe genérica.

NO ENTANTO, em JAVA é possível usar tipos primitivos, desde que empacotados como se fossem objetos, em um processo denominado *boxing*. Mais tarde, para ler os elementos deste mesmo *ArrayList*, estes são desempacotados (*unboxing*).

Resumindo

- *ArrayList* é indicado para criar listas de objetos!
- Mas o Java deu um jeitinho pra usarmos tipos primitivos também.

POO – *ArrayList*

Abaixo os principais métodos fornecidos por uma classe ArrayList:

- Método **add** – Adiciona um elemento no final do ArrayList.
- Método **clear** – Remove todos os elementos da estrutura.
- Método **contains** – Verifica se o ArrayList contém um elemento especificado, e retorna *true* em caso positivo, ou *false* caso contrário.
- Método **get** – Retorna o item em uma posição de índice especificada
- Método **indexOf** – Retorna a posição de índice da primeira ocorrência de um elemento especificado.
- Método **remove** – Remove a primeira ocorrência de um valor especificado ou de um elemento em um índice.
- Método **size** – Informa o número de elementos que estão armazenados na estrutura.
- Método **trimToSize** – Ajusta a capacidade do ArrayList de acordo com o número de elementos armazenados no momento.

POO - *ArrayList*

Vamos usar uma cópia simples da nossa classe de exemplo Aluno e Pessoa (camada Model) para construir exemplos simples de uso do *ArrayList*.

Vamos criar dentro de Principal (main class) uma lista simples de alunos.

Acompanhe o professor diretamente na IDE, o código fonte completo está nos anexos deste documento.

POO - *ArrayList*

```
package Model;
import java.util.*;


public class Aluno extends Pessoa {

    private String curso;
    private int fase;

    public Aluno() {
    }

    public Aluno(String curso, int fase) {
        this.curso = curso;
        this.fase = fase;
    }

    public Aluno(String curso, int fase, int id, String nome, int idade) {
        super(id, nome, idade);
        this.curso = curso;
        this.fase = fase;
    }

    public String getCurso() {
        return curso;
    }
    public void setCurso(String curso) {
        this.curso = curso;
    }
    public int getFase() {
        return fase;
    }
    public void setFase(int fase) {
        this.fase = fase;
    }

    // Override necessário para poder retornar os dados de Pessoa no toString para aluno.
    @Override
    public String toString() {
        return "\n ID: " + this.getId()
                + "\n Nome: " + this.getNome()
                + "\n Idade: " + this.getIdade()
                + "\n Curso: " + this.getCurso()
                + "\n Fase:" + this.getFase()
                + "\n ----------";
    }
}
```

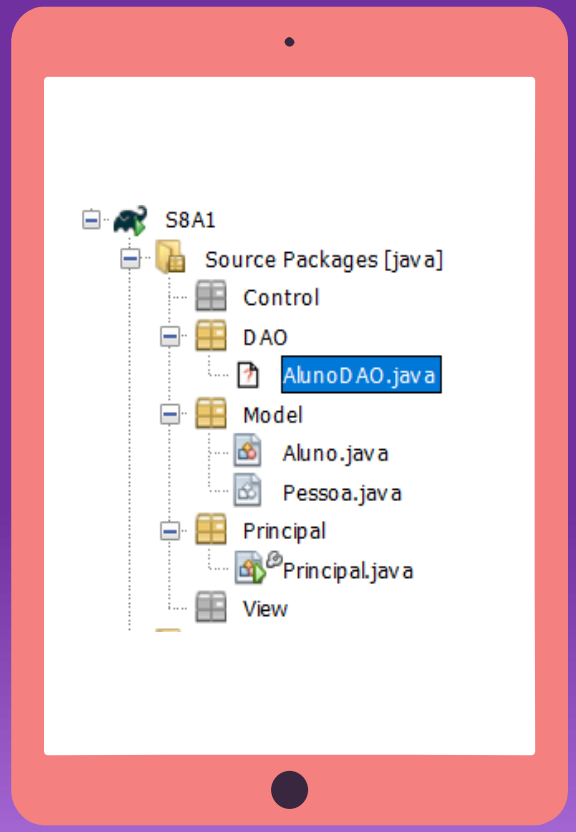
POO - *ArrayList*

```
import Model.Aluno;
import java.util.*;
public class Principal {
    public static void main(String[] args) {
        ArrayList<Aluno> listasimples = new ArrayList<>();

        // Criando objetos para inserir
        Aluno aluno1 = new Aluno("Sistemas", 7, 1111, "Tiburcio", 95);
        Aluno aluno2 = new Aluno("Massagem", 2, 2222, "Marilene", 21);
        listasimples.add(aluno1);
        listasimples.add(aluno2);
        System.out.println("\n ##########---------TESTE 1-------####### \n ");
        System.out.println(listasimples.toString());

        //apagando aluno1 da lista.
        listasimples.remove(0);
        System.out.println("\n ##########---------TESTE 2-------####### \n ");
        System.out.println(listasimples.toString());

        //alterando aluno2 da lista.
        Aluno aluno2alterado = new Aluno("Massagem Plus", 3, 2222, "Marileneeeee", 25);
        listasimples.set(0, aluno2alterado);
        System.out.println("\n ##########---------TESTE 3-------####### \n ");
        System.out.println(listasimples.toString());

// varredura da nossa listasimples listando só os nomes.
        Aluno aluno3 = new Aluno("Culinaria", 4, 3333, "Rita Lobo", 46);
        Aluno aluno4 = new Aluno("Culinaria", 8, 4444, "Claude Troisgros", 64);
        listasimples.add(aluno3);
        listasimples.add(aluno4);

        System.out.println("\n ##########---------TESTE 4-------####### \n ");
        int tamanholista = listasimples.size();
        for (int i = 0; i < tamanholista; i++) {
            System.out.println("Nome: " + listasimples.get(i).getNome());
        }

        // Usando Collections para ordenar
        Collections.sort(listasimples, new Comparator() {
            public int compare(Object o1, Object o2) {
                Aluno a1 = (Aluno) o1;
                Aluno a2 = (Aluno) o2;
                return a1.getNome().compareToIgnoreCase(a2.getNome());
                        // retorna -1 se for menor e +1 se for maior.
            }
        });

        System.out.println("\n ##########---------TESTE 5-------####### \n ");
        int tamanholista2 = listasimples.size();
        for (int i = 0; i < tamanholista2; i++) {
            System.out.println("Nome: " + listasimples.get(i).getNome());
        }
}}
```

POO – *ArrayList* simulando BD

Vamos ampliar mais um pouco nosso conceito de CAMADAS.

Agora vamos usar a camada DAO do nosso projeto para simular como ficariam os códigos e distribuição de classes se estivéssemos usando uma Base de Dados.



Os códigos a seguir já são **muito semelhantes as estruturas que utilizaremos** quando acoplarmos a conexão com **banco de dados**.

Descrição de imagem:
Mostra a árvore de diretórios do projeto, apenas sinalizando a existência de AlunoDao na camada DAO.

POO – *ArrayList* simulando BD

```
/*
 Aqui vamos simular a persistência de dados.
 Nas próximas aulas nós vamos reprogramar esta classe para conectar-se com o banco de dados.
 */

package DAO;

import Model.Aluno;
import java.util.*;

public class AlunoDAO {

    // vamos usar static para que a classe Aluno possa acessar SEM INSTANCIAR OBJETOS.
    public static ArrayList<Aluno> MinhaLista = new ArrayList<Aluno>();

}
```

POO – *ArrayList* simulando BD

Agora vamos criar novos métodos dentro da classe Aluno para que possamos simular a manipulação da nossa base de dados, que por enquanto, é um *ArrayList*.

Códigos completos nos anexos!

Acompanhe o professor demonstrando na IDE.

POO – *ArrayList* simulando BD

```
// DENTRO DA CLASSE ALUNO

    // Retorna a Lista de Alunos(objetos)
    public ArrayList getMinhaLista() {
        return AlunoDAO.MinhaLista;
    }

    // Cadastra novo aluno
    public boolean InsertAlunoBD(Aluno objeto) {
        AlunoDAO.MinhaLista.add(objeto);
        return true;
    }

    // Deleta um aluno específico pelo seu campo ID
    public boolean DeleteAlunoBD(int id) {
        int indice = this.procuraIndice(id);
        AlunoDAO.MinhaLista.remove(indice);
        return true;
    }

    // Edita um aluno específico pelo seu campo ID
    public boolean UpdateAlunoBD(int id, Aluno objeto) {
        int indice = this.procuraIndice(id);
        AlunoDAO.MinhaLista.set(indice, objeto);
        return true;
    }
```

```
// procura o ÍNDICE de objeto da MinhaLista que contem o ID enviado.
    private int procuraIndice(int id) {
        int indice = -1;
        for (int i = 0; i < AlunoDAO.MinhaLista.size(); i++) {
            if (AlunoDAO.MinhaLista.get(i).getId() == id) {
                indice = i;
            }
        }
        return indice;
    }
```

POO – *ArrayList* simulando DB

Neste momento temos:

- DAO.AlunoDAO.java
- Model.Pessoa.java
- Model.Aluno.java

Agora vamos usar esta estrutura.

Vamos programar dentro da nossa classe Principal, apenas para ilustrar o funcionamento, porém, a maioria dos códigos que veremos estarão dentro da **CAMADA CONTROL em uma aplicação REAL.**

POO – *ArrayList* simulando BD

// DENTRO DA CLASSE PRINCIPAL

```
// Cria objeto vazio de aluno apenas para manipular o métodos relacionados a MinhaLista
Aluno objetoaluno = new Aluno();

//Insere um objeto completo de Aluno em MinhaLista
objetoaluno.InsertAlunoBD(new Aluno("Sistemas", 7, 1111, "Tiburcio", 95));
System.out.println("\n ###########----------TESTE 1--------######## \n ");
System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
System.out.println(objetoaluno.getMinhaLista().toString());


// Insere um SEGUNDO objeto completo de Aluno em MinhaLista
objetoaluno.InsertAlunoBD(new Aluno("Massagem", 2, 2222, "Marilene", 21));
System.out.println("\n ###########----------TESTE 2--------######## \n ");
System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
System.out.println(objetoaluno.getMinhaLista().toString());

// APAGA um objeto de Aluno em MinhaLista Utilizando o campo ID como referência.
objetoaluno.DeleteAlunoBD(1111);
System.out.println("\n ###########----------TESTE 3--------######## \n ");
System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
System.out.println(objetoaluno.getMinhaLista().toString());

// EDITA um objeto de Aluno dentro de MinhaLista Utilizando o campo ID como referência e mandando outro objeto como modelo.
Aluno objetoaluno2 = new Aluno("Massagem Plus", 3, 2222, "Marileneeeee", 25);
objetoaluno.UpdateAlunoBD(2222, objetoaluno2);
System.out.println("\n ###########----------TESTE 4--------######## \n ");
System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
System.out.println(objetoaluno.getMinhaLista().toString());
```

POO – PILHA (*STACK*)

A classe java.util.Stack implementa uma pilha, segue o critério (**LIFO - Last In First Out**).

Principais operações

1. boolean *empty()*: Se a lista está vazia.
2. E *peek()*: Olha o objeto que está no topo da pilha, sem apagá-.lo.
3. E *pop()*:Remove o objeto do topo da pilha e retorna o objeto.
4. E *push(E item)*:Coloca um objeto no topo da pilha.
5. int *search(Object obj)*: Retorna a posição que um objeto está na pilha.

Descrição de imagem:
Mostra foto de livros empilhados.

POO – PILHA (*STACK*)

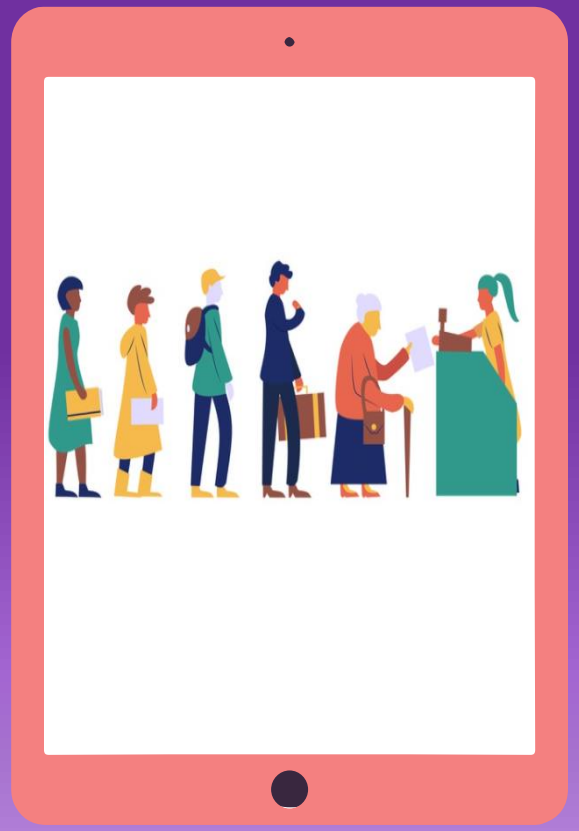
```
package Principal;
import Model.Aluno;
import java.util.*;

public class TestesPilha {

    public static void main(String[] args) {


        Stack<Aluno> pilha = new Stack<Aluno>();
        pilha.push(new Aluno("Sistemas", 7, 1111, "Tiburcio", 95));
        pilha.push(new Aluno("Massagem", 2, 2222, "Marilene", 21));
        pilha.push(new Aluno("Culinaria", 4, 3333, "Rita Lobo", 46));
        pilha.push(new Aluno("Culinaria", 8, 4444, "Claude Troisgros", 64));

        System.out.println("Topo:" + pilha.peek().getId() + " - " + pilha.peek().getNome());

        // Interator utilizado para correr a pilha, veja mais na documentação JAVA
        for (Iterator it = pilha.iterator(); it.hasNext();) {
            Aluno c = (Aluno) it.next();
            System.out.println(c.getId() + " - " + c.getNome());
        }

        System.out.println("Removendo:" + pilha.peek().getNome());
        pilha.pop();

        System.out.println("Topo:" + pilha.peek().getNome());
    }
}
```

POO – FILA (*QUEUE*)

A classe java.util.Queue implementa uma fila, segue o critério (***FIFO - First In First Out***).

Principais operações

1. boolean *offer(E item)* : Coloca o objeto no fim da fila.
2. E *poll()* : Remove o objeto do início da fila e retorna o objeto ou retorna null se vazio.
3. E *peek()* : Olha o objeto que está no início da fila, sem apagar ou retorna null se vazio.
4. boolean *add(E item)* : Coloca o objeto no fim da fila.
5. E *remove()* : Remove o objeto do início da fila e retorna o objeto.
6. E *element()* : Olha o objeto que está no início da fila, sem apagar.



Descrição de imagem:
Mostra foto de uma fila de pessoas.

POO – FILA *(QUEUE)*

```
package Principal;
import Model.Aluno;
import java.util.*;

public class TesteFila {

    public static void main(String[] args) {

        Queue<Aluno> fila = new LinkedList();
        fila.offer(new Aluno("Sistemas", 7, 1111, "Tiburcio", 95));
        fila.offer(new Aluno("Massagem", 2, 2222, "Marilene", 21));
        fila.offer(new Aluno("Culinaria", 4, 3333, "Rita Lobo", 46));
        fila.offer(new Aluno("Culinaria", 8, 4444, "Claude Troisgros", 64));

        System.out.println("Inicio da Fila:" + fila.peek().getId() + " - " + fila.peek().getNome());

        // Interator utilizado para correr a fila, veja mais na documentação JAVA
        for (Iterator it = fila.iterator(); it.hasNext();) {
            Aluno c = (Aluno) it.next();
            System.out.println(c.getId() + " - " + c.getNome());
        }

        System.out.println("Removendo:" + fila.peek().getNome());
        fila.poll();

        System.out.println("Topo:" + fila.peek().getNome());
    }
}
```

EXERCÍCIO 34

Modifique o exemplo de PILHA da aula passada, agora peça os dados para o usuário.

Experimente os outros métodos disponíveis.

EXERCÍCIO 35

Modifique o exemplo de FILA da aula passada, agora peça os dados para o usuário.

Experimente os outros métodos disponíveis.

EXERCÍCIO 36

Criar um exemplo em JAVA usando ArrayList para uma lista de professores, use as classes das aulas anteriores.

ânima EDUCAÇÃO

UNISUL
UNIVERSIDADE DO SUL DE SANTA CATARINA

FIM

ANEXO 1 – DAO.AlunoDAO.java

```
/*
 Aqui vamos simular a persistência de dados.
 Nas próximas aulas nós vamos reprogramar esta classe para conectar-se com o banco de dados.
 */
package DAO;

import Model.Aluno;
import java.util.*;

public class AlunoDAO {

    public static ArrayList<Aluno> MinhaLista = new ArrayList<Aluno>();

}
```

ANEXO 2 – Model.Pessoa.java

```
package Model;

public abstract class Pessoa {

    // Atributos
    private int id;
    private String nome;
    private int idade;

    // Método Construtor de Objeto Vazio
    public Pessoa() {
    }

    // Método Construtor de Objeto, inserindo dados
    public Pessoa(int id, String nome, int idade) {
        this.id = id;
        this.nome = nome;
        this.idade = idade;
    }

    // Métodos GET e SET
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

}
```

ANEXO 3 – Model.Aluno.java

```
package Model;

import java.util.*;
import DAO.AlunoDAO;

public class Aluno extends Pessoa {

    // Atributos
    private String curso;
    private int fase;

    // Método Construtor de Objeto Vazio
    public Aluno() {
    }

    // Método Construtor de Objeto, inserindo dados
    public Aluno(String curso, int fase) {
        this.curso = curso;
        this.fase = fase;
    }

    // Método Construtor usando também o construtor da SUPERCLASSE
    public Aluno(String curso, int fase, int id, String nome, int idade) {
        super(id, nome, idade);
        this.curso = curso;
        this.fase = fase;
    }

    // Métodos GET e SET
    public String getCurso() {
        return curso;
    }

    public void setCurso(String curso) {
        this.curso = curso;
    }

    public int getFase() {
        return fase;
    }

    public void setFase(int fase) {
        this.fase = fase;
    }

    // Override necessário para poder retornar os dados de
    // Pessoa no toString para aluno.
    @Override
    public String toString() {
        return "\n ID: " + this.getId()
                + "\n Nome: " + this.getNome()
                + "\n Idade: " + this.getIdade()
                + "\n Curso: " + this.getCurso()
                + "\n Fase:" + this.getFase()
                + "\n ----------";    }

     /*    ABAIXO OS MÉTODOS PARA USO JUNTO COM O DAO
        SIMULANDO A ESTRUTURA EM CAMADAS PARA
        USAR COM BANCOS DE DADOS.     */

    // Retorna a Lista de Alunos(objetos)
    public ArrayList getMinhaLista() {
        return AlunoDAO.MinhaLista;
    }

    // Cadastra novo aluno
    public boolean InsertAlunoBD(Aluno objeto) {
        AlunoDAO.MinhaLista.add(objeto);
        return true;
    }

    // Deleta um aluno específico pelo seu campo ID
        public boolean DeleteAlunoBD(int id) {
        int indice = this.procuraIndice(id);
        AlunoDAO.MinhaLista.remove(indice);
        return true;
    }

    // Edita um aluno específico pelo seu campo ID
    public boolean UpdateAlunoBD(int id, Aluno objeto) {
        int indice = this.procuraIndice(id);
        AlunoDAO.MinhaLista.set(indice, objeto);
        return true;
    }

    // procura o INDICE de objeto da MinhaLista que contem o ID enviado.
    private int procuraIndice(int id) {
        int indice = -1;
        for (int i = 0; i < AlunoDAO.MinhaLista.size(); i++) {
            if (AlunoDAO.MinhaLista.get(i).getId() == id) {
                indice = i;
            }
        }
        return indice;
    }
}
```

ANEXO 4 – Principal.java (testes simples de ArrayList)

```
package Principal;

import DAO.AlunoDAO;
import Model.Aluno;
import java.util.*;

public class Principal {

    public static void main(String[] args) {

        /*
 SIMULAÇÃO SIMPLES DO USO DE ARRAYLIST
         */
        ArrayList<Aluno> listasimples = new ArrayList<>();

        // Criando objetos para inserir
        Aluno aluno1 = new Aluno("Sistemas", 7, 1111, "Tiburcio", 95);
        Aluno aluno2 = new Aluno("Massagem", 2, 2222, "Marilene", 21);

        listasimples.add(aluno1);
        listasimples.add(aluno2);

        System.out.println("\n ##########----------TESTE 1--------####### \n ");
        System.out.println(listasimples.toString());

        //apagando aluno1 da lista.
        listasimples.remove(0);

        System.out.println("\n ##########----------TESTE 2--------####### \n ");
        System.out.println(listasimples.toString());

        Aluno aluno2alterado = new Aluno("Massagem Plus", 3, 2222, "Marileneeeee", 25);
        listasimples.set(0, aluno2alterado);
        System.out.println("\n ##########----------TESTE 3--------####### \n ");
        System.out.println(listasimples.toString());

        // varredura da nossa listasimples listando só os nomes.
        Aluno aluno3 = new Aluno("Culinaria", 4, 3333, "Rita Lobo", 46);
        Aluno aluno4 = new Aluno("Culinaria", 8, 4444, "Claude Troisgros", 64);

        listasimples.add(aluno3);
        listasimples.add(aluno4);

        System.out.println("\n ##########----------TESTE 4--------####### \n ");
        int tamanholista = listasimples.size();
        for (int i = 0; i < tamanholista; i++) {
            System.out.println("Nome: " + listasimples.get(i).getNome());
        }


        // Usando Collections para ordenar
        Collections.sort(listasimples, new Comparator() {

            public int compare(Object o1, Object o2) {
                Aluno a1 = (Aluno) o1;
                Aluno a2 = (Aluno) o2;
                return a1.getNome().compareToIgnoreCase(a2.getNome()); // retorna -1 se for menor e +1 se for maior.
            }
        });


        System.out.println("\n ##########----------TESTE 5--------####### \n ");
        int tamanholista2 = listasimples.size();
        for (int i = 0; i < tamanholista2; i++) {
            System.out.println("Nome: " + listasimples.get(i).getNome());
        }


    }
}
```

ANEXO 5 – Principal.java (testes usando DAO)

```
package Principal;

import DAO.AlunoDAO;
import Model.Aluno;
import java.util.*;

public class Principal {

    public static void main(String[] args) {

 /*
    SIMULAÇÃO USANDO DAO (NESTE CASO O NOSSA BD É UM ARRAYLIST )
       */

       // Cria objeto vazio de aluno apenas para manipular o métodos relacionados a MinhaLista
       Aluno objetoaluno = new Aluno();

       //Insere um objeto completo de Aluno em MinhaLista
       objetoaluno.InsertAlunoBD(new Aluno("Sistemas", 7, 1111, "Tiburcio", 95));
       System.out.println("\n ##########----------TESTE 1--------######## \n ");
       System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
       System.out.println(objetoaluno.getMinhaLista().toString());

       //Insere um SEGUNDO objeto completo de Aluno em MinhaLista
       objetoaluno.InsertAlunoBD(new Aluno("Massagem", 2, 2222, "Marilene", 21));
       System.out.println("\n ##########----------TESTE 2--------######## \n ");
       System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
       System.out.println(objetoaluno.getMinhaLista().toString());

       // APAGA um objeto de Aluno em MinhaLista Utilizando o campo ID como referência.
       objetoaluno.DeleteAlunoBD(1111);
       System.out.println("\n ##########----------TESTE 3--------######## \n ");
       System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
       System.out.println(objetoaluno.getMinhaLista().toString());

       // EDITA um objeto de Aluno dentro de MinhaLista Utilizando o campo ID como referência e mandando outro objeto como modelo.
       Aluno objetoaluno2 = new Aluno("Massagem Plus", 3, 2222, "Marileneeeee", 25);
       objetoaluno.UpdateAlunoBD(2222, objetoaluno2);
       System.out.println("\n ##########----------TESTE 4--------######## \n ");
       System.out.println("Tamanho da lista: " + objetoaluno.getMinhaLista().size());
       System.out.println(objetoaluno.getMinhaLista().toString());

    }
}
```