transforme-se

serasa experian

WISEHANDS
Tecnologia e Acessibilidade

JAVA
AULA 7 - Tipo Enum e as Collections

WISEHANDS
Tecnologia e Acessibilidade

Tipo Enum


serasa
experian


WH
WISEHANDS
Tecnologia e Acessibilidade

Enum

São tipos de campos que consistem em um conjunto fixo de constantes (static final), sendo como uma lista de valores pré-definidos. Na linguagem de programação Java, pode ser definido um tipo de enumeração usando a palavra chave enum.

```
1 | public enum Turno  {
2 |     MANHA, TARDE, NOITE;
3 | }
```

Enum

Por serem os campos de uma enum constantes, seus nomes são escritos em letras maiúsculas.
No exemplo acima temos três campos que correspondem aos turnos manhã, tarde e noite.

Para atribuir um desses valores a uma variável podemos fazer como no código abaixo:

```
1 | Turno  turno = Turno.MANHA;
```

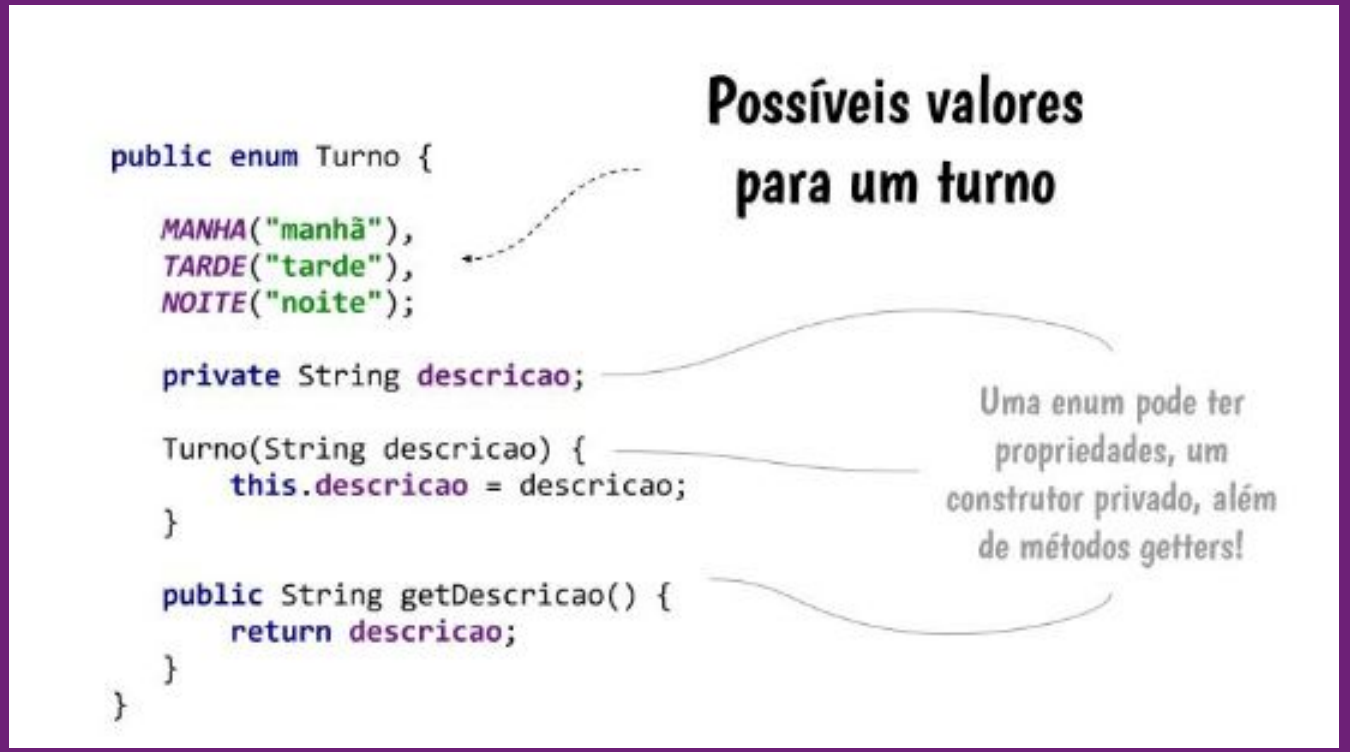
Perceba que ao utilizar enums limitamos os valores que podem ser atribuídos a uma variável.

Sendo assim, devemos atribuir ao campo Turno um dos valores pré-definidos na enum Turno.

Classes enum e herança

Ao declarar uma enum estamos implicitamente estendendo a classe java.lang.Enum. Isso cria algumas limitações, porque o Java não suporta herança múltipla, o que impede uma classe enum de estender outras classes.

Porém, uma classe enum pode ter propriedades, assim como um construtor e um método.



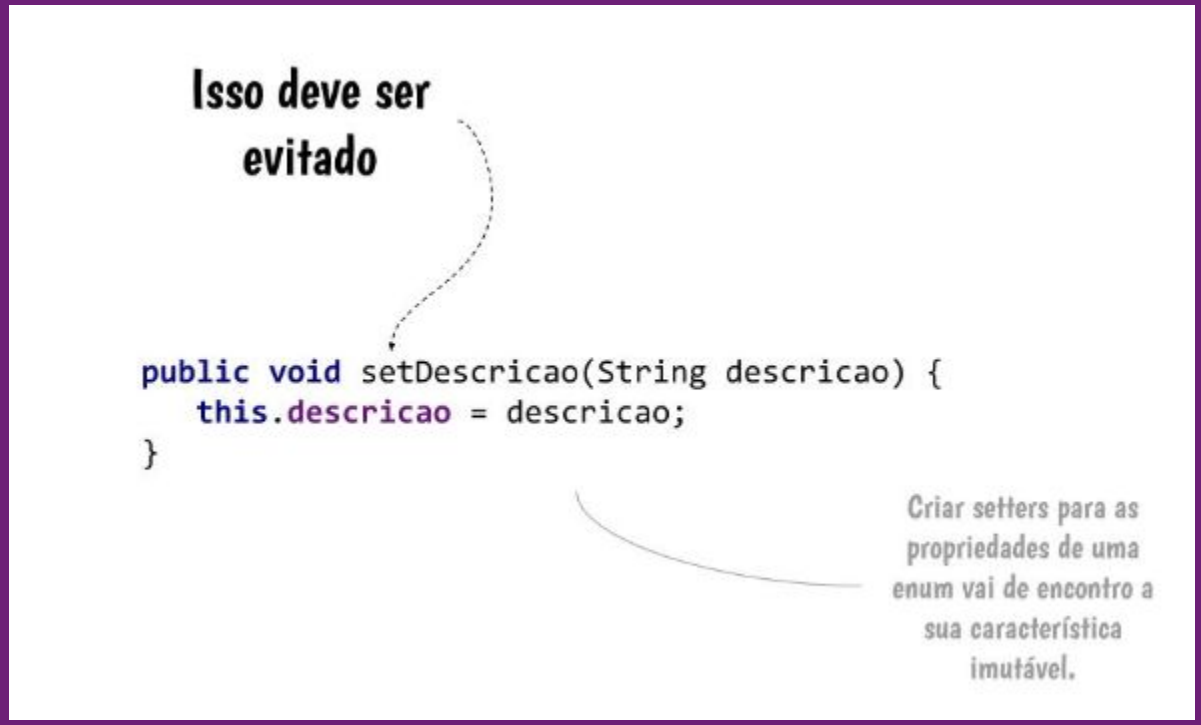
Classes enum e herança

O código abaixo apresenta uma segunda versão da enum Turno, agora com a propriedade descrição, um construtor, que inicia essa propriedade, além de um método para retornar essa descrição:

```java
public enum Turno {

    MANHA("manhã"),
    TARDE("tarde"),
    NOITE("noite");

    private String descricao;

    Turno(String descricao) {
        this.descricao = descricao;
    }

    public String getDescricao() {
        return descricao;
    }
}
```

Classes enum e herança

O construtor de uma enum é sempre privado, não podendo ser invocado diretamente. Nele são iniciados todos os campos, que por serem constantes devem declarados antes das propriedades e do construtor da classe.



Métodos herdados de Enum

Uma vez que Turno herda de Enum, possui métodos declarados na classe pai. Um dos mais utilizados dentre eles é values(), que retorna um array contendo todos os campos da enum.

No exemplo a seguir imprimimos a descrição de cada um dos campos da enum Turno:

```
1 | for (Turno t : Turno.values()) {
2 |         System.out.println(t.getDescricao());
3 |   }
```

Métodos herdados de Enum

Outros métodos úteis dessa classe são:

Método	**Retorno**	**Descrição**
toString()	String	Retorna uma String com o nome da instância (em maiúsculas).
valueOf(String nome)	static <T extends Enum<T>> T	Retorna o objeto da classe enum cujo nome é a string do argumento.
ordinal()	int	Retorna o número de ordem do objeto na enumeração.

Características dos tipos enum

- As instâncias dos tipos enum são criadas e nomeadas junto com a declaração da classe, sendo fixas e imutáveis (o valor é fixo);
- Não é permitido criar novas instâncias com a palavra chave new;
- O construtor é declarado private, embora não precise de modificador private explícito;
- Seguindo a convenção, por serem objetos constantes e imutáveis (static final), os nomes declarados recebem todas as letras em MAIÚSCULAS;
- As instâncias dos tipos enum devem obrigatoriamente ter apenas um nome;
- Opcionalmente, a declaração da classe pode incluir variáveis de instância, construtor, métodos de instância, de classe, etc.

Collections

serasa experian

WISEHANDS
Tecnologia e Acessibilidade

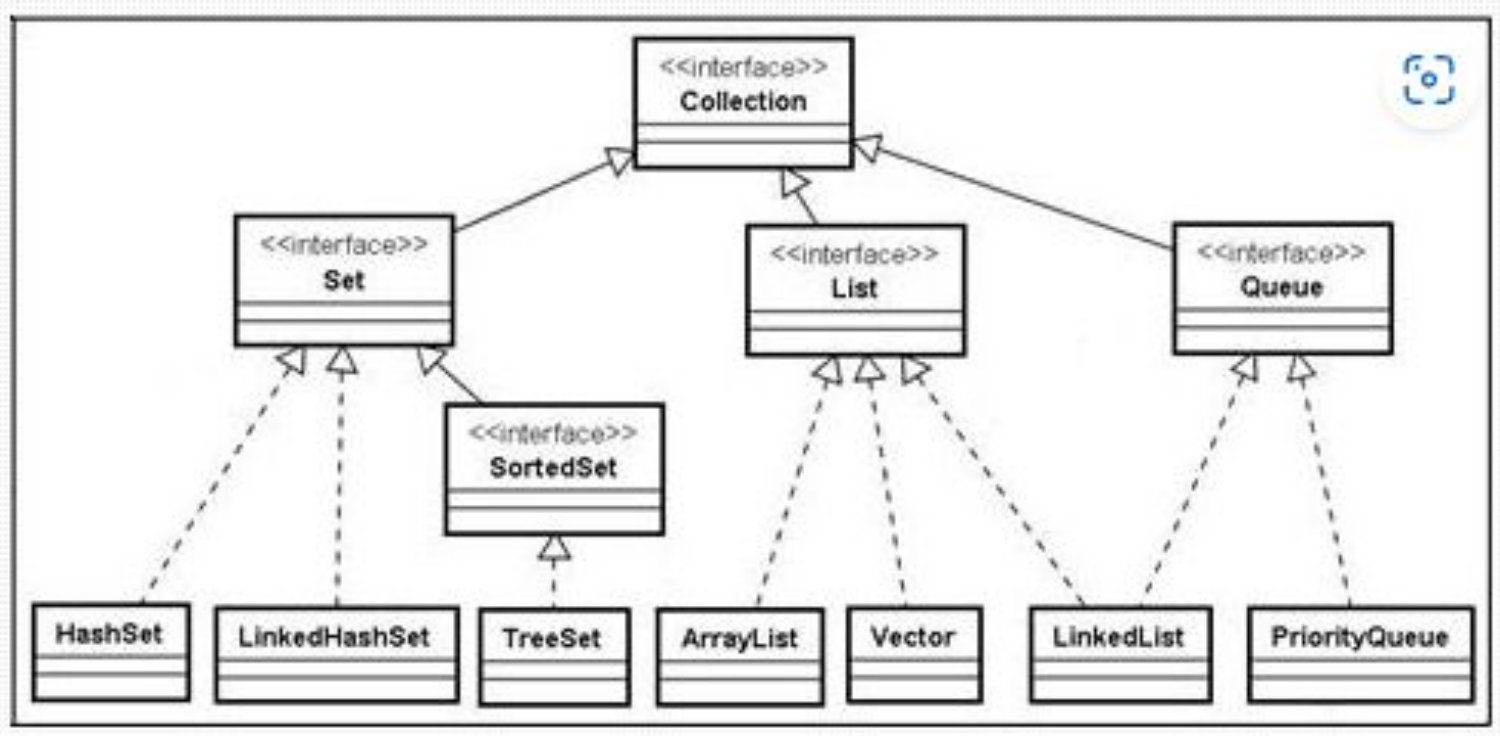
Collections - O que é Collections Framework?

Collections Framework é um conjunto bem definido de interfaces e classes para representar e tratar grupos de dados como uma única unidade, que pode ser chamada coleção, ou collection. A Collections Framework contém os seguintes elementos:

- **Interfaces**: tipos abstratos que representam as coleções. Permitem que coleções sejam manipuladas tendo como base o conceito “Programar para interfaces e não para implementações”, desde que o acesso aos objetos se restrinja apenas ao uso de métodos definidos nas interfaces;
- **Implementações**: são as implementações concretas das interfaces;
- **Algoritmos**: são os métodos que realizam as operações sobre os objetos das coleções, tais como busca e ordenação.

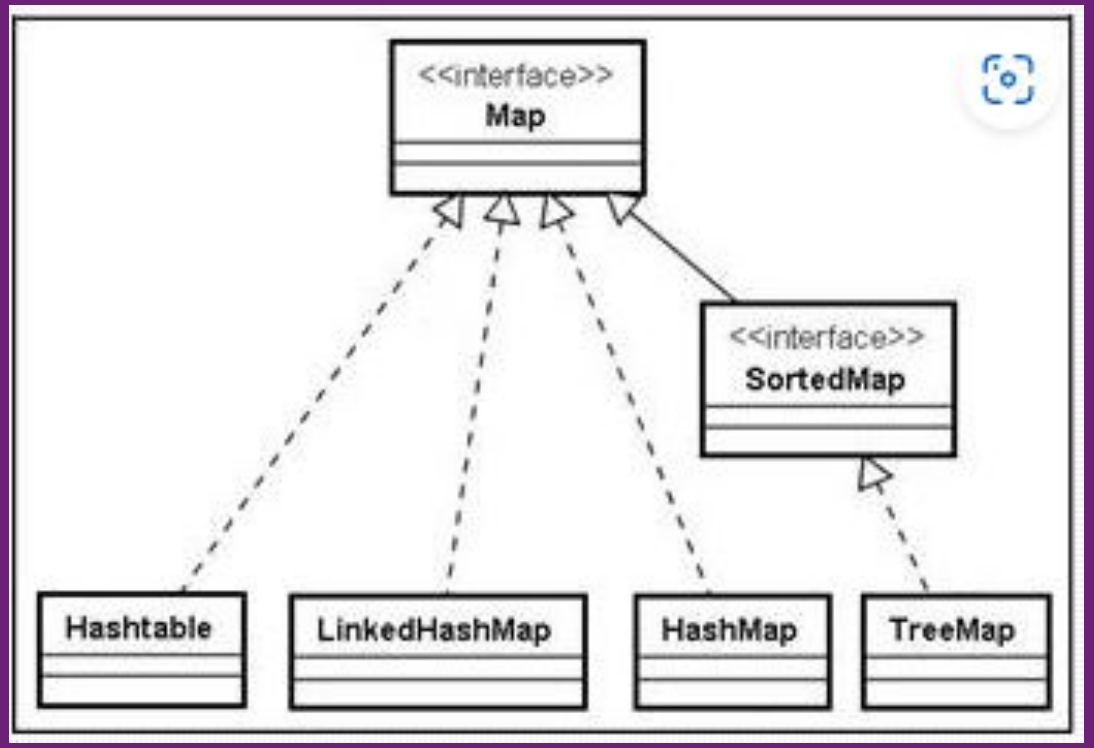
Collections - O que é Collections Framework?

- A Figura mostra a árvore da hierarquia de interfaces e classes da Java Collections Framework que são derivadas da interface Collection. O diagrama usa a notação da UML, onde as linhas cheias representam extends e as linhas pontilhadas representam implements.



Collections - O que é Collections Framework?

A hierarquia da Collections Framework tem uma segunda árvore. São as classes e interfaces relacionadas a mapas, que não são derivadas de Collection, como mostra a Figura. Essas interfaces, mesmo não sendo consideradas coleções, podem ser manipuladas como tal.



Collection`s Interfaces

Neste momento vamos apresentar uma breve descrição de cada uma das interfaces da hierarquia:

Collection – está no topo da hierarquia. Não existe implementação direta dessa interface, mas ela define as operações básicas para as coleções, como adicionar, remover, esvaziar, etc.;

Set – interface que define uma coleção que não permite elementos duplicados. A interface SortedSet, que estende Set, possibilita a classificação natural dos elementos, tal como a ordem alfabética;

List – define uma coleção ordenada, podendo conter elementos duplicados. Em geral, o usuário tem controle total sobre a posição onde cada elemento é inserido e pode recuperá-los através de seus índices. Prefira esta interface quando precisar de acesso aleatório, através do índice do elemento;

Collection`s Interfaces

Neste momento vamos apresentar uma breve descrição de cada uma das interfaces da hierarquia:

Queue – um tipo de coleção para manter uma lista de prioridades, onde a ordem dos seus elementos, definida pela implementação de Comparable ou Comparator, determina essa prioridade. Com a interface fila pode-se criar filas e pilhas;

Map – mapeia chaves para valores. Cada elemento tem na verdade dois objetos: uma chave e um valor. Valores podem ser duplicados, mas chaves não. SortedMap é uma interface que estende Map, e permite classificação ascendente das chaves. Uma aplicação dessa interface é a classe Properties, que é usada para persistir propriedades/configurações de um sistema, por exemplo.

Collection`s Implementações

As interfaces apresentadas anteriormente possuem diversas implementações que são utilizadas para armazenar as coleções. Abaixo estão resumidas as implementações mais comuns.

ArrayList – é como um array cujo tamanho pode crescer. A busca de um elemento é rápida, mas inserções e exclusões não são. Podemos afirmar que as inserções e exclusões são lineares, o tempo cresce com o aumento do tamanho da estrutura. Esta implementação é preferível quando se deseja acesso mais rápido aos elementos. Por exemplo, se você quiser criar um catálogo com os livros de sua biblioteca pessoal e cada obra inserida receber um número sequencial (que será usado para acesso) a partir de zero;

LinkedList – implementa uma lista ligada, ou seja, cada nó contém o dado e uma referência para o próximo nó. Ao contrário do ArrayList, a busca é linear e inserções e exclusões são rápidas. Portanto, prefira LinkedList quando a aplicação exigir grande quantidade de inserções e exclusões. Um pequeno sistema para gerenciar suas compras mensais de supermercado pode ser uma boa aplicação, dada a necessidade de constantes inclusões e exclusões de produtos;

Collection`s Implementações

As interfaces apresentadas anteriormente possuem diversas implementações que são utilizadas para armazenar as coleções. Abaixo estão resumidas as implementações mais comuns.

HashSet – o acesso aos dados é mais rápido que em um TreeSet, mas nada garante que os dados estarão ordenados. Escolha este conjunto quando a solução exigir elementos únicos e a ordem não for importante. Poderíamos usar esta implementação para criar um catálogo pessoal das canções da nossa discografia;

TreeSet – os dados são classificados, mas o acesso é mais lento que em um HashSet. Se a necessidade for um conjunto com elementos não duplicados e acesso em ordem natural, prefira o TreeSet. É recomendado utilizar esta coleção para as mesmas aplicações de HashSet, com a vantagem dos objetos estarem em ordem natural;

Collection`s Implementações

As interfaces apresentadas anteriormente possuem diversas implementações que são utilizadas para armazenar as coleções. Abaixo estão resumidas as implementações mais comuns.

LinkedHashSet – é derivada de HashSet, mas mantém uma lista duplamente ligada através de seus itens. Seus elementos são iterados na ordem em que foram inseridos. Opcionalmente é possível criar um LinkedHashSet que seja percorrido na ordem em que os elementos foram acessados na última iteração. Poderíamos usar esta implementação para registrar a chegada dos corredores de uma maratona;

HashMap – baseada em tabela de espalhamento, permite chaves e valores null. Não existe garantia que os dados ficarão ordenados. Escolha esta implementação se a ordenação não for importante e desejar uma estrutura onde seja necessário um ID (identificador). Um exemplo de aplicação é o catálogo da biblioteca pessoal, onde a chave poderia ser o ISBN (International Standard Book Number);

Collection`s Implementações

As interfaces apresentadas anteriormente possuem diversas implementações que são utilizadas para armazenar as coleções. Abaixo estão resumidas as implementações mais comuns.

TreeMap – implementa a interface SortedMap. Há garantia que o mapa estará classificado em ordem ascendente das chaves. Mas existe a opção de especificar uma ordem diferente. Use esta implementação para um mapa ordenado. Aplicação semelhante a HashMap, com a diferença que TreeMap perde no quesito desempenho;

LinkedHashMap – mantém uma lista duplamente ligada através de seus itens. A ordem de iteração é a ordem em que as chaves são inseridas no mapa. Se for necessário um mapa onde os elementos são iterados na ordem em que foram inseridos, use esta implementação. O registro dos corredores de uma maratona, onde a chave seria o número que cada um recebe no ato da inscrição, é um exemplo de aplicação desta coleção.

Collection`s Implementações

Cada uma das implementações tem todos os métodos definidos em suas interfaces. Em qualquer uma delas é possível inserir elementos null. Em mapas, tanto chaves quanto valores podem ser null.

A interface List

List tem duas implementações – ArrayList e LinkedList. ArrayList oferece acesso aleatório rápido através do índice. Já em LinkedList o acesso aleatório é lento e necessita de um objeto nó para cada elemento, que é composto pelo dado propriamente dito e uma referência para o próximo nó, ou seja, consome mais memória. Além dessas considerações, se for necessário inserir elementos no início e deletar elementos no interior da lista, a melhor opção poderia ser LinkedList.

```java
import java.util.*;

public class ListaAluno {

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

    Aluno a = new Aluno("João da Silva", "Linux básico", 0);
    Aluno b = new Aluno("Antonio Sousa", "OpenOffice", 0);
    Aluno c = new Aluno("Lúcia Ferreira", "Internet", 0);
    Aluno d = new Aluno("Antonio Sousa", "OpenOffice", 0);
    lista.add(a);
    lista.add(b);
    lista.add(c);
    lista.add(d);
    System.out.println(lista);
    Aluno aluno;
    Iterator<Aluno> itr = lista.iterator();
    while (itr.hasNext()) {
      aluno = itr.next();
      System.out.println(aluno.getNome());
    }
  }

}
```

A interface Set

Uma das características de List é que ela permite elementos duplicados, o que não é desejável em nossa lista de alunos. Analisando as interfaces, concluímos que Set é o que realmente precisamos, pois não permite elementos duplicados. Como HashSet tem desempenho superior a TreeSet, optamos por esta implementação.

```java
import java.util.*;

public class ListaAluno {

  public static void main(String[] args) {
    Set<Aluno> conjunto = new HashSet<Aluno>();

    Aluno a = new Aluno("João da Silva", "Linux básico", 0);
    Aluno b = new Aluno("Antonio Sousa", "OpenOffice", 0);
    Aluno c = new Aluno("Lúcia Ferreira", "Internet", 0);
    Aluno d = new Aluno("Antonio Sousa", "OpenOffice", 0);
    conjunto.add(a);
    conjunto.add(b);
    conjunto.add(c);
    conjunto.add(d);
    System.out.println(conjunto);
  }
}
```

A interface Map

Vamos supor que agora queremos uma estrutura onde possamos recuperar os dados de um aluno passando apenas o seu nome como argumento de um método. Ou seja, informamos o nome do aluno e o objeto correspondente a esse nome é devolvido. Para isso vamos usar a interface Map, que não estende Collection. Isso causa uma mudança profunda na aplicação, visto que os métodos usados anteriormente não poderão ser usados. Map tem seus próprios métodos para inserir/buscar/remover elementos na estrutura.

Esta interface mapeia chaves para valores. Considerando a nova proposta do problema, a chave será o nome do aluno e o valor será o objeto aluno.

Para usar uma classe que implementa Map, quaisquer classes que forem utilizadas como chave devem sobrescrever os métodos hashCode() e equals(). Isso é necessário porque em um Map as chaves não podem ser duplicadas, apesar dos valores poderem ser.D

A interface Map

```java
import java.util.*;

public class MapaAluno {

  public static void main(String[] args) {
    Map<String, Aluno> mapa = new TreeMap<String, Aluno>();

    Aluno a = new Aluno("João da Silva", "Linux básico", 0);
    Aluno b = new Aluno("Antonio Sousa", "OpenOffice", 0);
    Aluno c = new Aluno("Lúcia Ferreira", "Internet", 0);
    Aluno d = new Aluno("Benedito Silva", "OpenOffice", 0);
    mapa.put("João da Silva", a);
    mapa.put("Antonio Sousa", b);
    mapa.put("Lúcia Ferreira", c);
    mapa.put("Benedito Silva", d);
    System.out.println(mapa);
    System.out.println(mapa.get("Lúcia Ferreira"));

    Collection<Aluno> alunos = mapa.values();
    for (Aluno e : alunos) {
      System.out.println(e);
    }
  }

}
```

transforme-se

O conhecimento é o poder
de transformar o seu futuro.