iscte
INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

Módulo 1: Programação em Linguagem Java

Aula 11
Enumerados, Interfaces e Comparadores

UPskill

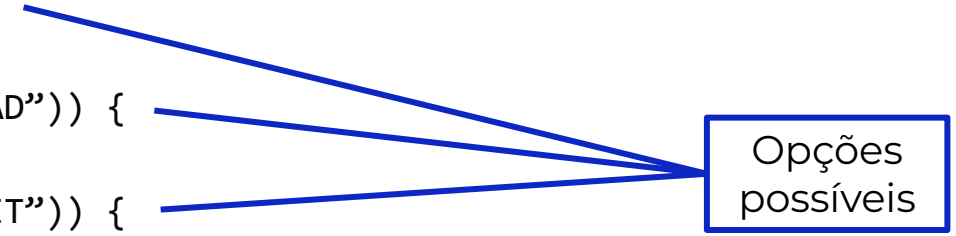
Enumerados

O que são **Enumerados**

- Tipo com conjunto fixo e finito de valores
    - Exemplos: dias da semana, direções, estado civil
- Podem ter atributos e construtores
- Têm operações associadas
- Melhores que inteiros ou cadeias de caracteres para representar pequenos conjuntos

Sem enumerados: **Opções de Menu**

```
Scanner scanner = new Scanner(System.in);
System.out.println("Introduza um comando:");
String command = scanner.nextLine();
if(command.equals("SAVE")) {
   // gravar...
} else if(command.equals("LOAD")) {
   // carregar...
} else if(command.equals("EXIT")) {
   // sair...
}
```



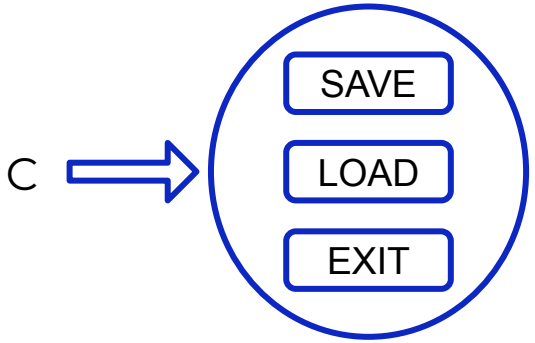
Com enumerados: **Opções de Menu**

```
public enum Command { SAVE, LOAD, EXIT; }

Scanner scanner = new Scanner(System.in);
System.out.println("Introduza um comando:");
String line = scanner.nextLine();
Command command = Command.valueOf(line);
if(command == Command.SAVE) {
   // gravar...
} else if(command == Command.LOAD) {
   // carregar...
} else if(command == Command.EXIT) {
   // sair...
}
```

Operação valueOf()

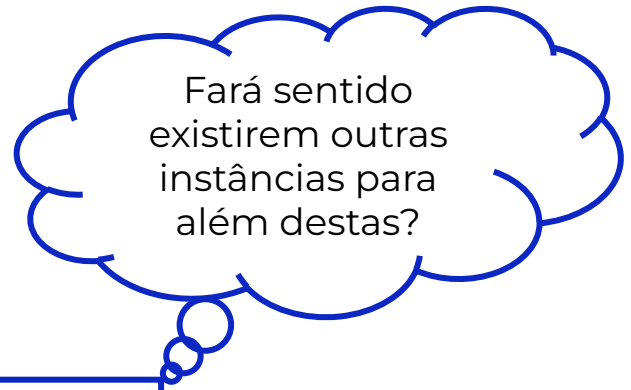
- Disponível em todos os tipos de enumerados
- Obtém o elemento de um enumerado dado o seu nome (`String`)

```
public enum Command {
   SAVE, LOAD, EXIT;
}

Command c = Command.valueOf(“LOAD”);
```



Sem enumerados: **Direções**

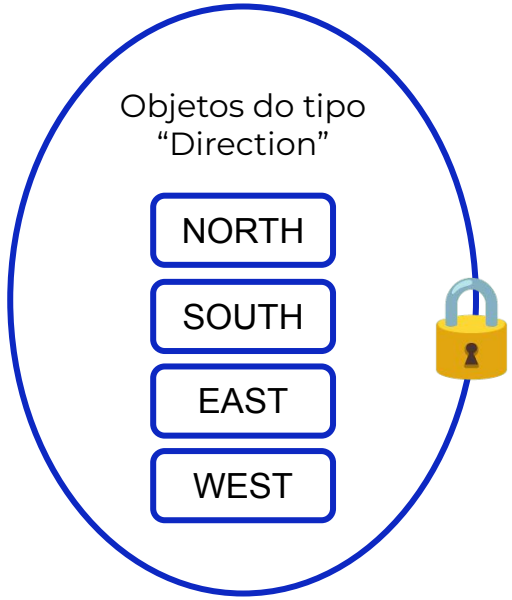
```
public class Direction {
    private String name;
    public Direction(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```




```
Direction north = new Direction(“North”);
Direction south = new Direction(“South”);
Direction east = new Direction(“East”);
Direction west = new Direction(“West”);
```

Com enumerados: **Direções**

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST;
    public String prettyName() {
        return name().charAt(0) + name().substring(1).toLowerCase();
    }
}

//...

String s1 = Direction.NORTH.name();
System.out.println(s1);
String s2 = Direction.SOUTH.prettyName();
System.out.println(s2);
```



Operação name()

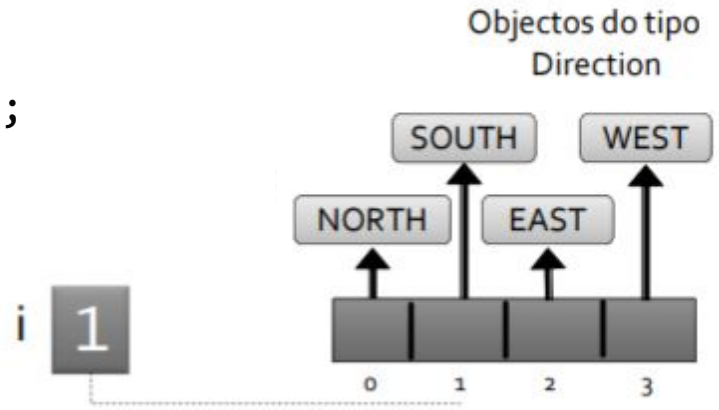
- Disponível em todos os tipos enumerados
- Devolve um objeto String com o identificador do elemento do enumerado

```
String s = Direction.WEST.name();
```

Operação ordinal()

- Disponível em todos os tipos enumerados
- Devolve o índice do elemento do enumerado de acordo com a ordem de declaração

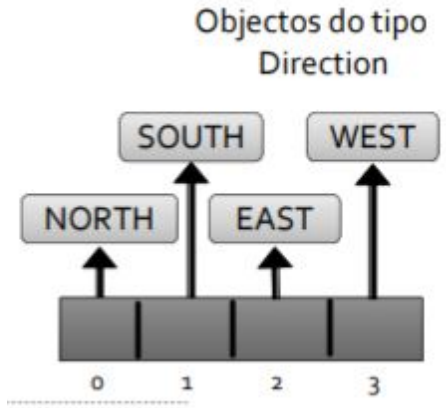
```
int i = Direction.SOUTH.ordinal();
```



Operação values()

- Disponível em todos os tipos enumerados
- Devolve um vector com todos os elementos do enumerado (pela ordem que são declarados)

```
Direction[] dirs = Direction.values();
```



Exercício 1

- Escreva um enumerado que represente as quatro operações matemáticas: somar, subtrair, multiplicar e dividir.

- Desenvolva um método que realize todas as operações do enumerado.

Interfaces

Interfaces

- Definem um comportamento que as classes podem implementar.
- Contêm apenas constantes e declarações de métodos.
- Uma interface não é uma classe.

```
public interface NomeInterface1 {

    ...
}
```

- Uma classe pode implementar várias interfaces.
- Uma interface pode estender várias interfaces.
- Se uma classe implementar uma interface **é necessário que todos os métodos da interface sejam definidos na classe**.

Interfaces

- Esta relação é especificada no cabeçalho da classe que assume implementar uma determinada interface através da palavra reservada implements.

```
public class NomeClass implements NomeInterface1 {
    ...
}
```

- Caso a classe implemente mais do que uma interface basta incluir os respectivos nomes separados por vírgulas.

```
public class NomeClass implements NomeInterface1, NomeInterface2 {
    ...
}
```

Exemplo de **Interface**

```
public interface FiguraGeometricaPlana {
    public String getNomeFiguraPlana();
    public int getArea();
    public int getPerimetro();
}
```

Apenas é definida a assinatura. A implementação só é feita nas classes que implementam a interface.

As Interfaces definem **um modelo de comportamento, mas não a sua implementação.**

Implementação da **Interface**

```
public class Quadrado implements FiguraGeometricaPlana{
    private int lado;
    public int getLado(){
        return lado;
    }
    public void setLado(){
        this.lado=lado;
    }

    @Override
    public int getArea(){
        return lado * lado;
    }
    @Override
    public int getPerimetro(){
        int perimetro = lado * 4;
        return perimetro;
    }

    @Override
    public String getNomeFiguraPlana(){
        return "Quadrado";
    }
}
```

Implementou-se a interface.
Esta classe é agora forçada a implementar os comportamentos que a interface prevê.

Interfaces vs **Classes Abstratas**

- Métodos da interface são **implicitamente abstratos.** Uma classe abstrata no entanto pode ter métodos implementados por defeito.
- Métodos das interfaces são **públicas** por defeito. Uma classe abstrata pode ser privada, protected, etc.
- A interface deve ser implementada com a chave **“implements”**, em vez do **“extends”**.
- Interfaces só podem estender outras interfaces
- As classes (abstratas inclusive) podem estender uma única classe, mas implementar várias interfaces

Exercício 2

1. Crie as classes `Rectangulo` e `Circulo` que implementam a interface FiguraGeometricaPlana, como foi feito para a classe Quadrado.

2. Implemente a interface `FiguraGeometricaTridimensional` em todas as classes. O nome de todas as figuras tridimensionais irá ser prismas ou cilindro.

```
public interface FiguraGeometricaTridimensional {
   public String getNomeFiguraTridimensinal();
   public int getAltura();
   public int getVolume();
}
```

O Java inclui já um conjunto de **interfaces úteis**.

Comparadores

Comparable

- A interface `Comparable` é usada para **ordenar objectos** de uma classe definida pelo utilizador.
- Esta interface tem apenas um método denominado `compareTo(Object)`.
- O método é usado para comparar o objecto actual (`this`) com o objeto passado por argumento à função.
- Retorna: Inteiro positivo, se o objeto atual for maior que o objeto especificado. Inteiro negativo, se o objeto actual for menor que o objeto especificado. Zero, se o objeto for igual ao objeto especificado.

Quando implementar o **Comparable?**

- A interface Comparable deve ser implementada sempre que pretendemos definir uma **ordem natural** (intrinseca) para o objeto em causa.
- Estamos a definir um **comparador interno**.
- Esta ordenação é **especialmente útil ao guardar os objetos em listas ou outras estruturas de dados** (mais à frente no curso).
- Mais à frente nesta aula vamos descobrir uma outra forma de fazer uma comparação entre objetos.

Exemplo de **Comparable**

```
public class Estudante implements Comparable<Estudante>{
   private int numero;
   private String nome;
   private int idade;

   public  Estudante(int numero,String nome,int idade){
       this.numero=numero;
       this.nome=nome;
       this.idade=idade;
   }

   public int compareTo(Estudante st){
       if(idade==st.idade)
           return 0;
       else if(idade>st.getIdade())
           return 1;
       else
           return-1;
   }
      ...
}
```

Estamos a definir a ordem natural de comparação **entre dois objetos da classe Estudante**.

Exemplo 2

```
public static void main(String [] args){
   Estudante [] st = new Estudante[3];
   st[0]=new Estudante(101,”José”,23);
   st[1]=new Estudante(106,”Maria”,27);
   st[2]=new Estudante(105,”Manuel”,21);

   Arrays.sort(st);

   System.out.println(“Ordenado por idade”);

   for(int i=0; i<st.length; i++){
       System.out.println(st[i].getNumero() + ” “ + st[i].getNome() + ” ” +
st[i].getIdade());
   }
}
```

Esta instrução permite ordenar os objetos do array **pela sua ordem natural**.

Ou seja, precisamos de definir o comparable para isto funcionar.

Resultado na consola:

```
Ordenado por idade
105 Manuel 21
101 José 23
106 Maria 27
```

Exercício 3

Crie a classe `Filme` que vai ter os seguintes atributos: pontuação, nome e ano. Defina a ordem natural da classe `Filme` de acordo com os seguintes critérios:

1. Ordene os seguintes filmes por ordem crescente do seu ano;
2. Ordene os seguintes filmes por ordem decrescente em relação à sua pontuação.

Filmes:

- Force Awakens, 8, 2015
- Star Wars, 9, 1977
- Empire Strikes Back, 7, 1980
- Return of the Jedi, 10, 1983

Comparator

- Outra forma de definir a comparação entre dois objetos.
- A interface `Comparator` é usada para ordenar objectos de uma classe definida pelo utilizador.
- Esta interface tem o método `compare(Object obj1, Object obj2)`.
- Ao contrário do Comparable, o `Comparator` é **externo ao tipo de objecto** que está a ser comparado.
- Podem ser criadas classes separadas (que implementam o Comparator) para fazer comparações segundo diferentes critérios:
    - Exemplo: `AccountComparatorByDate, AccountComparatorByCountry`

Exemplo **Comparator**

```
public class NumeroCompare implements Comparator<Estudante> {
    public int compare(Estudante s1, Estudante s2){
        if (s1.getNumero() < s2.getNumero()) {
            return -1;
        } else if (s1.getNumero() > s2.getNumero()) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Estamos a definir uma **nova classe**, que compara os estudantes por número.

Note-se que este comparador **é externo à classe Estudante**.

Exemplo **Comparator**

```
NomeCompare nomeCompare = new NomeCompare();
Arrays.sort(st, nomeCompare);
System.out.println("Ordenado por nome");
for(int i=0; i<estudantes.length; i++){
   System.out.println(stu[i].getNumero() + " " + stu[i].getNome() + stu[i].getIdade());
}

NumeroCompare numCompare=new NumeroCompare();
Arrays.sort(st, numCompare);
System.out.println("Ordenado por numero");
for(int i=0; i<estudantes.length; i++){
   System.out.println(stu[i].getNumero() + " " + stu[i].getNome() + stu[i].getIdade());
}
```

Também podemos fazer a ordenação passando por argumento um comparador.

Ou seja, podemos usar o sort() **sem argumentos para ordenar pela ordem natural** (o objeto precisa de ser comparable), ou **passar um comparador para ordenar por um critério específico**.

```
Ordenado por nome
106 Maria 27
105 Manuel 21
101 José 23

Ordenado por numero
101 José 23
105 Manuel 21
106 Maria 27
```

Exercício 4

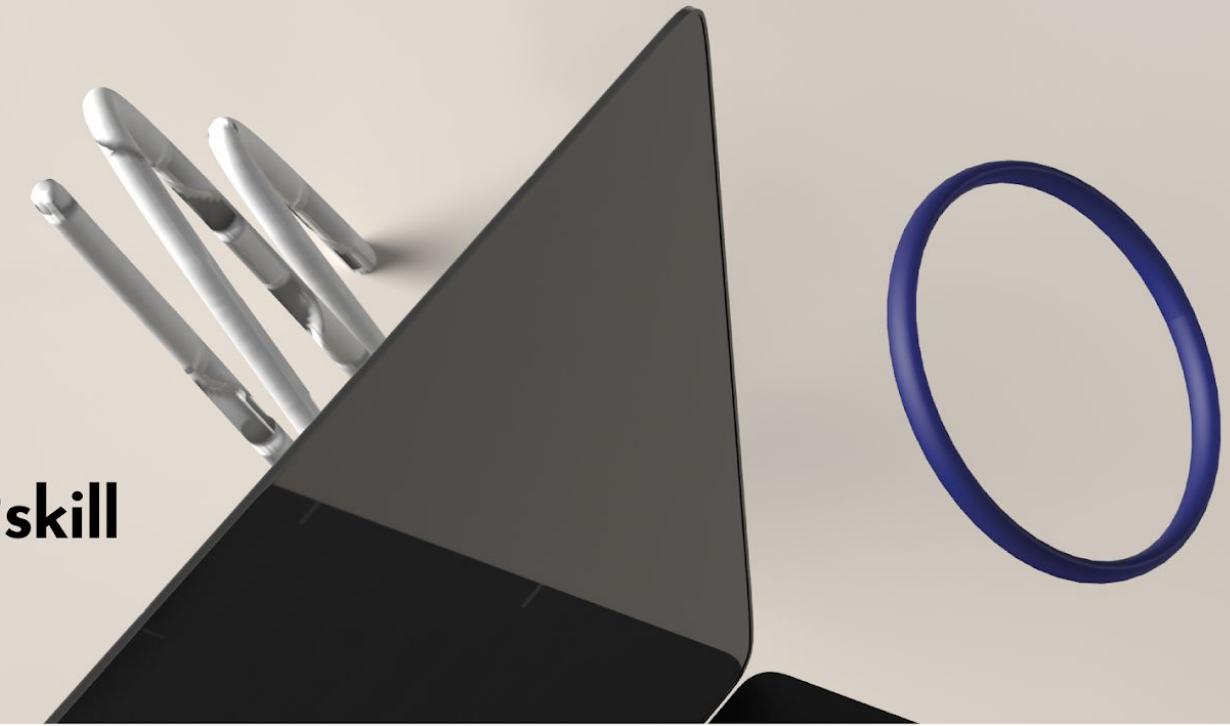
Crie a classe `Livro` que tem os seguintes atributos: nome, ano, editora e autor. Implemente 3 comparadores externos que permitem ordenar os livros por:

1) Ordem alfabética crescente do título
2) Ordem decrescente do ano
3) Ordem alfabética crescente da editora

Livros:

- The Shinning, 1990, Leya, Stephen King
- Harry Potter e a Pedra Filosofal, 2001, ASA, Joanne Rowling
- O Inferno de Dante, 1472, Bertrand, Dante
- Conde de Monte Cristo, 1844, Texto, Alexandre Dumas
- Desenhar Bases de Dados, 2016, Silabo, Pedro Nogueira

O futuro profissional começa aqui



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

UPskill