

# Java

## Tipos Enumerados

UA, DETI, Programação III  
José Luis Oliveira, Carlos Costa  
2017/18

1

## Enumerados

- Um Enumerado é uma forma simples de associar constantes a um conjunto de valores.
- Antes do JAVA 5:

```
public static final int SPRING = 0;  
public static final int SUMMER = 1;  
public static final int FALL = 2;  
public static final int WINTER = 3;
```

Este formato tem um problema!!!

Qual?

Como se resolve?

2

## Exemplo: Solução 1

```
class Note {
    private int value;
    private Note(int val) { value = val; }
    public static final Note
        MIDDLE_C = new Note(0),
        C_SHARP = new Note(1),
        C_FLAT = new Note(2);
}
class Instrument {
    public void play(Note n) {
        System.out.println("Instrument.play()");
    }
}
class Wind extends Instrument {
    public void play(Note n) {
        System.out.println("Wind.play()");
    }
}
public class Music {
    public static void tune(Instrument i) { // ...
        i.play(Note.MIDDLE_C);
    }
    public static void main(String[] args) {
        Wind flute = new Wind();
        tune(flute); // Upcasting
    }
}
```

```
class Note {
    public static final int
        MIDDLE_C = 0,
        C_SHARP = 1,
        C_FLAT = 2;
}
```

--> Play(int n)  
Não cria qq.  
ligação a Note

3

## Exemplo: Solução 2, usando enum

```
enum Note { MIDDLE_C, C_SHARP, C_FLAT }

class Instrument {
    public void play(Note n) {
        System.out.println("Instrument.play()");
    }
}
class Wind extends Instrument {
    public void play(Note n) {
        System.out.println("Wind.play()");
    }
}
public class Music {
    public static void tune(Instrument i) {
        i.play(Note.MIDDLE_C);
    }
    public static void main(String[] args) {
        Wind flute = new Wind();
        tune(flute); // Upcasting
    }
}
```

```
enum Note {
    MIDDLE_C(0),
    C_SHARP(1),
    C_FLAT(2);
    private int value;
    private Note(int val) {
        value = val;
    }
}
```

4

## Tipos Enumerados

- Mais Valia Importante: “compile-time type safety”

- Forma mais Simples

```
public enum Color { WHITE, BLACK, RED, YELLOW, BLUE  
}
```

- Forma de referenciar

```
Color.WHITE, Color.RED, etc
```

- Dentro de uma Classe

```
public class Externa{  
    public enum Color {    WHITE, BLACK, RED, YELLOW, BLUE  
    }  
}
```

- Forma de referenciar

```
Externa.Color.WHITE, Externa.Color.RED, etc
```

5

## Tipos Enumerados em JAVA

- **enum** é uma *classe*, não um tipo primitivo.
  - São Objectos - podemos utilizar em Collections;
  - Pode implementar uma Interface.
  - Suportam comparação (== ou equals()).
- Tipos enumerados **não são inteiros**.
- Só têm **construtores privados**.
- Os **valores enumerados** são automaticamente **public, static, final**.

6

## Enum - uma Classe

- Podemos ter tipos Enumerados com dados e operações associadas:

```
public enum Color {  
    WHITE(21), BLACK(22), RED(23), YELLOW(24), BLUE(25);  
  
    // Dados  
    private int code;  
  
    // Construtor  
    private Color(int c) {    code = c; }  
  
    // Método  
    public int getCode() {    return code; }  
}
```

7

## Enum - implements Interface

- Os tipos enum podem implementar Interfaces

```
public enum Color implements Runnable {  
    WHITE, BLACK, RED, YELLOW, BLUE;  
  
    public void run() {  
        System.out.println("name()=" + name() + ",  
                           toString()=" + toString());  
    }  
}
```

- Utilização:  
for(Color c : Color.values()) { c.run(); }
- Ou  
for(Runnable r : Color.values()) { r.run(); }

8

## Enum - Métodos Disponíveis

- São Comparable (têm uma ordem).
- Fornecem alguns métodos úteis:
  - toString()
  - valueOf(String val) : converte a String (elemento do conjunto) para um valor
  - ordinal(): posição (int) do valor na lista de elementos
  - values(): devolve a lista de elementos

9

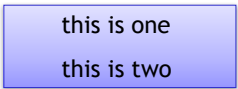
## Enum - toString

- Por omissão, a representação tipo String é o próprio nome de cada elemento. No entanto, podemos modificar redefinindo o método toString().

```
public enum MyType {  
    ONE {  
        public String toString() {return "this is one";}  
    },  
  
    TWO {  
        public String toString() {return "this is two";}  
    }  
}
```

- Main:

```
public class EnumTest {  
    public static void main(String[] args){  
        System.out.println(MyType.ONE);  
        System.out.println(MyType.TWO);  
    }  
}
```



this is one  
this is two

10

## Enum - values()

- O método “values()” retorna um array com todos os elementos.

```
Name[] nameValues = Name.values();
```

- Exemplo de Utilização:

```
for (Name n : Name.values()) {  
    // ... ;  
}
```

11

## Enum - Switch Statement

- A instrução switch funciona com enumerados

```
Color myColor = Color.BLACK;  
  
switch (myColor) {  
    case WHITE: ...;  
    case BLACK: ...;  
    ...  
    case BLUE: ...;  
    default: ...;  
}
```

12

## Enum - ordinal()

- Definimos um Enum para os Meses do Ano

```
public enum Mes {  
    JANEIRO, FEVEREIRO, MARCO, ABRIL, MAIO,  
    JUNHO, JULHO, AGOSTO, SETEMBRO, OUTUBRO,  
    NOVENBRO, DEZEMBRO ;  
}
```

- Se utilizarmos na classe Data

```
// Construtor de Data  
public Data (int iDia, Mes iMes, int iAno){  
    //...  
}  
// Main  
Data d1 = new Data(11, Mes.MAIO, 1900);
```

13

## Exemplo 1

```
enum Mes {  
    JANEIRO, FEVEREIRO, MARCO, ABRIL,  
    MAIO, JUNHO, JULHO, AGOSTO,  
    SETEMBRO, OUTUBRO, NOVENBRO, DEZEMBRO ;  
}  
  
public class Enum1 {  
    public static void main(String[] args) {  
        for (Mes t: Mes.values())  
            System.out.println(t+" : "+t.ordinal());  
    }  
}
```

```
JANEIRO, JANEIRO : 0  
FEVEREIRO, FEVEREIRO : 1  
MARCO, MARCO : 2  
ABRIL, ABRIL : 3  
...
```

14

```
enum Mes {
    JANEIRO(1), FEVEREIRO(2), MARCO(3), ABRIL(4),
    MAIO(5), JUNHO(6), JULHO(7), AGOSTO(8),
    SETEMBRO(9), OUTUBRO(10), NOVEMBRO(11), DEZEMBRO(12);

    private final int mes;
    private Mes(int m) {
        this.mes=m;
    }
    public int numMes() {
        return mes;
    }
    @Override public String toString() {
        return this.name().substring(0, 1)+
            (this.name().substring(1,this.name().length())).toLowerCase();
    }
}

public class Enum2 {
    public static void main(String[] args) {
        for (Mes t: Mes.values())
            System.out.println(t+" : "+
                t.ordinal() + " , "+t.numMes());
    }
}
```

```
Janeiro, JANEIRO : 0, 1
Fevereiro, FEVEREIRO : 1, 2
...
```

## Exemplo 3

```
public enum Ensemble {
    SOLO(1), DUET(2), TRIO(3), QUARTET(4), QUINTET(5),
    SEXTET(6), SEPTET(7), OCTET(8), DOUBLE_QUARTET(8),
    NONET(9), DECTET(10), TRIPLE_QUARTET(12);

    private final int numberOfMusicians;
    Ensemble(int size) {
        numberOfMusicians = size;
    }

    public int numberOfMusicians() {
        return numberOfMusicians;
    }
}
```



```

public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27, 7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7),
    PLUTO   (1.27e+22, 1.137e6);

    private final double mass;    // in kilograms
    private final double radius; // in meters
    Planet(double mass, double radius) {
        this.mass = mass;    this.radius = radius;
    }
    public double mass()    { return mass; }
    public double radius() { return radius; }

    // universal gravitational constant (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    public double surfaceGravity() {return G*mass/(radius*radius); }
    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}

```