

# Tipos Enumerados

- [Interesse](#)
- [Definição](#)
- [Implementação](#)
- [Declaração](#)
- [Valor de um Tipo Enumerado](#)
- [Variável de Tipo Enumerado](#)
- Métodos Automáticos
  - [name](#)
  - [toString](#)
  - [values](#)
  - [compareTo](#)
  - [equals](#)
  - [valueOf](#)
  - [ordinal](#)
- [Métodos Programados](#)

■ **Representação Computacional**

- Conjuntos **fixos** de **constantes**
  - Constantes **Enumeradas** // relevante **ordem** das constantes
  - Constantes **Não-Enumeradas** // irrelevante ordem das constantes

■ **Exemplos Típicos**

- Constantes **Enumeradas**
  - Dias da semana: { Segunda, Terça, Quarta, Quinta, Sexta, Sábado, Domingo }
  - Meses do ano: { Janeiro, Fevereiro, Março, Abril, Maio, Junho, Julho, Agosto, Setembro, Outubro, Novembro, Dezembro }
  - Tipos de Cartas: { Duque, Terno, Quadra, Quina, Sena, Oito, Nove, Dez, Dama, Valete, Rei, Manilha, As }
  - Notas Escolares: { Mau, Medíocre, Suficiente, Bom, Muito Bom }
- Constantes **Não-Enumeradas**
  - Conjunto de cores: { Branco, Preto, Azul }
  - Pontos cardeais: { Norte, Sul, Este, Oeste }
  - Naipes de cartas: { Copas, Espadas, Ouros, Paus }

## ▪ Tipo Enumerado

- É um tipo de dados
- Definido pelo utilizador
- Valores são **constantes** ... **enumeradas** automaticamente desde **zero**

Constante 1	Constante 2	Constante 3	...	Constante N	Valores do <b>tipo</b>
0	1	2	...	N-1	<b>Ordem</b>


## ▪ Característica Importante de um Tipo Enumerado

- Estabelece **ordem** nos seus valores
  - Valor 1 < Valor 2 < Valor 3 < ... < Valor N
- Exemplo
  - Interessa aos Conjuntos
    - Dias da Semana = { Segunda, Terça, Quarta, Quinta, Sexta, Sábado, Domingo }
    - Meses = { Janeiro, Fevereiro, Março, Abril, Maio, Junho, Julho, Agosto, Setembro, Outubro, Novembro, Dezembro }

- Pode ser implementado numa classe especial // desde Java5

- Chamada **enum**
- Não instanciável
- Possuindo

```
public enum Cor {  
    AMARELO, AZUL, BRANCO, ENCARNADO  
}
```



- Valores do Tipo // Ex: \_\_\_\_\_
- Métodos Automáticos // adicionados **automaticamente** pelo compilador
  - name // para obter **nome** de 1 valor
  - toString // para obter **descrição** textual de 1 valor
  - values // para obter **array** contendo todos os valores do tipo enumerado
  - compareTo // para determinar **ordem relativa** de 2 valores ( v1 <, > ou = v2)
  - equals // para testar **igualdade** de 2 valores
  - valueOf // para converter **nome** de valor (string) no **valor** correspondente
  - ordinal // para obter **ordem** (posição) de 1 valor
- Métodos Programados
  - Métodos de Instância
  - Métodos de Classe
- Variáveis
  - Instância
  - Classe

- Herda implicitamente a classe **Enum**
  - public **abstract** class **Enum**< E extends Enum<E> > **implements** Comparable<E>, Serializable { ... }
  - package java.lang

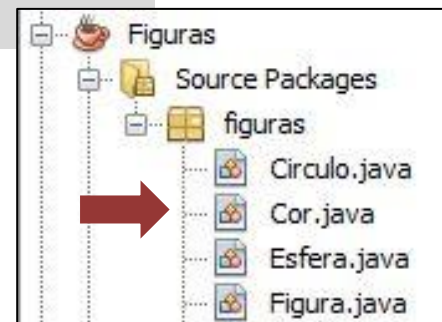
## ■ Pode ser

- Classe Independente // Dentro de package

```
public enum Cor {  
    AMARELO, AZUL, BRANCO, ENCARNADO  
}
```

Netbeans

1. Clique direito na package
2. New Java Enum



- Classe Interna // Dentro de classes e interfaces

```
public class Figura {  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
}
```

```
public interface FigurasGeometricas {  
    enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
}
```

Constantes de Interfaces

- Por omissão
  - public static final

## ▪ Sintaxe

```
[modificador de acesso] enum nome {  
    constante1, constante2, ..., constanteN  
}
```

[...] significa opcional

- Modificador de acesso
  - public
  - package
- nome
  - Deve ter letra **inicial** maiúscula
- constantes
  - Podem ter letras maiúsculas e/ou minúsculas
  - Enumeradas **automaticamente** desde 0 pelo compilador

## ▪ Exemplos

```
public enum Cor {  
    AMARELO, AZUL, BRANCO, ENCARNADO  
}
```

Constantes enumeradas desde 0  
AMARELO=0 ; AZUL=1 ; BRANCO=2 ; ENCARNADO=3

```
public enum Cor {  
    Amarelo, Azul, Branco, Encarnado  
}
```

- Como atributo
- Sintaxe

```
[modificador de acesso] [static] enum nome { constante1, constante2, ..., constanteN }
```


- Modificador de acesso: private, package, protected ou public
- **nome** deve ter letra inicial maiúscula
- **constantes** podem ter letras maiúsculas e/ou minúsculas

- Exemplo

```
public class Exemplo {  
    ...  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO } // public ⇒ acesso exterior  
    ...  
}
```

- Proibida num método

```
public class Exemplo {  
    ...  
    public void metodo() {  
        enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
    }  
}
```





## ▪ Referir um valor

## ▪ Caso Geral

## ▪ Sintaxe

tipo\_enumerado.constante\_enumerada

// prefixo → operador\_ponto → nome\_constante

## ▪ Exemplos

```
Cor.ENCARNADO;           // indica valor ENCARNADO do tipo Cor
Cor.AMARELO;              // indica valor AMARELO do tipo Cor
```

## ▪ Caso Particular

## ▪ Tipo enumerado declarado como classe interna

## ▪ Indicação no exterior da classe hospedeira

## ▪ Sintaxe

Classe\_hospedeira.tipo\_enumerado.constante\_enumerada

## ▪ Exemplos

```
Exemplo.Cor.ENCARNADO;    // indica valor ENCARNADO do tipo Exemplo.Cor
Exemplo.Cor.AMARELO;      // indica valor AMARELO do tipo Exemplo.Cor
```

```
public class Exemplo {
    public static enum Cor { ... }
    // public ⇒ acesso exterior
    ...
}
```

## ▪ Declaração

## ▪ Sintaxe

tipo\_enumerado nome\_variável [= valor\_do\_tipo\_enumerado];

## ▪ Exemplo

```
public class Exemplo {  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
    ...  
    public void metodo() {  
        Cor c;                // inicializada a null, por omissão  
        ...  
    }  
}
```

## ▪ Só pode guardar valores do tipo enumerado declarado

## ▪ Exemplo

- Variável `c` só pode armazenar valores do tipo enumerado `Cor`

```
Cor c = Cor.ENCARNADO;  
c = Cor.BRANCO;
```

## ▪ Interesse

- Obter **nome** de um valor

## ▪ Declaração

- `public String name() { ... }`      `// método de instância`

## ▪ Retorna

- Nome do valor sobre o qual é aplicado o método

```
public class Exemplo {  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
    public void metodo() {  
        // método name aplicado a valor  
        String nome = Cor.ENCARNADO.name();           // nome = ENCARNADO  
        ...  
        // método aplicado a variável de tipo enumerado  
        Cor c1 = Cor.BRANCO;                          // variável c1 do tipo enumerado Cor  
        System.out.println( c1.name() );               // imprime BRANCO  
    }  
}
```

## ▪ Interesse

- Obter **descrição** textual de um valor

## ▪ Declaração

- `public String toString() { ... }` // método de instância

## ▪ Retorna

- Por omissão (i.e. se não for redefinido)
  - Semelhante ao método name
  - Retorna nome do valor sobre o qual é aplicado o método

```
public class Exemplo {  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
    public void metodo() {  
        // método toString aplicado a valor  
        String descricao = Cor.ENCARNADO.toString(); // descricao = ENCARNADO, por omissão  
        ...  
        // método aplicado a variável de tipo enumerado  
        Cor c1 = Cor.BRANCO; // variável c1 do tipo enumerado Cor  
        System.out.println( c1.toString() ); // imprime BRANCO  
    }  
}
```

- Pode ser redefinido para cada valor

- Exemplo

```
public class Exemplo {  
    public static enum DiasDaSemana {  
        SABADO { public String toString() { return "Sábado"; } },           // redefinição toString()  
        DOMINGO { public String toString() { return "Domingo"; } },  
        SEGUNDA { public String toString() { return "Segunda-feira"; } },  
        TERCA { public String toString() { return "Terça-feira"; } },  
        QUARTA { public String toString() { return "Quarta-feira"; } },  
        QUINTA { public String toString() { return "Quinta-feira"; } },  
        SEXTA { public String toString() { return "Sexta-feira"; } }  
    }  
  
    public void metodo() {  
        String dia = DiasDaSemana.SEGUNDA.toString();           // dia = "Segunda-feira"  
    }  
}
```

- **Interesse**
  - Obter **array** contendo todos os valores do tipo enumerado
    - Exemplo: varrimento de todos os valores do tipo enumerado
- **Declaração**
  - `public static tipoEnumerado[] values() { ... }`
- **Método de classe (static)**
  - Aplica-se ao tipo enumerado
    - Exemplo: `Cor.values()`
- **Retorna**
  - Array com todos os valores do tipo enumerado sobre o qual é aplicado o método

```
public class Exemplo {  
  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
  
    public void metodo() {  
        ...  
        // Imprimir todas as descrições de valores do tipo Cor  
        for( Cor c : Cor.values() )  
            System.out.println( c );           // por omissão: c.toString()  
    }  
}
```

## ▪ Interesse

- Determinar **posição relativa** de 2 valores do mesmo tipo enumerado

## ▪ Declaração

- `public int compareTo(TipoEnumeradoDoRecetor o){ ... }` // tipo enum implementa **Comparable**

## ▪ Retorna Valor Inteiro

// método de instância

- -1 // valor1 < valor2
- 0 // valor1 = valor2
- 1 // valor1 > valor2

Exemplo  
`valor1.compareTo( valor2 )`

```
public class Exemplo {  
    public static enum Nota { MAU, MEDIOCRE, SUFICIENTE, BOM, MUITO BOM }  
    public void metodo() {  
        Nota n1 = ... , n2 = ...;           // Ex: Nota n1 = Nota.BOM, n2 = Nota.MAU;  
        if ( n1.compareTo(n2) > 0 )  
            System.out.println( n1.name() + " é melhor do que " + n2.name() );  
        else if ( n1.compareTo(n2) < 0 )  
            System.out.println(" n1.name() + " é pior do que " + n2.name() );  
        else  
            System.out.println(" Notas iguais ");  
    }  
}
```

## ▪ Interesse

- Testar a igualdade de 2 valores

## ▪ Declaração

- `public boolean equals(Object o)` // método de instância

## ▪ Retorna

- Resultado do teste de igualdade entre *Object* `o` e *valor* sobre o qual é aplicado o método
  - `true` ou `false`

```
public class Exemplo {  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
    public void metodo() {  
        ArrayList figuras = new ArrayList();           // para guardar instâncias de figuras geométricas  
        ...  
        for( Object obj : figuras){  
            Figura f = (Figura) obj;  
            if( f.getCor().equals( Cor.ENCARNADO ) )    // Equivalente : f.getCor() == Cor.ENCARNADO  
                System.out.println( f );              // imprime só figuras de cor ENCARNADO  
        }  
    }  
}
```



## ▪ Interesse

- Converter **nome** dum valor no **valor** correspondente
- Exemplo `"BRANCO" → Cor.BRANCO`

## ▪ Declaração

- `public static tipo_enumerado valueOf(String name)`

## ▪ Método de classe (static)

- Aplicado ao tipo enumerado
- Exemplo `Cor.valueOf ("BRANCO")`

## ▪ Retorna

- Valor com nome passado por parâmetro
- Exemplo

```
Cor c = Cor.valueOf ("BRANCO");           // c = Cor.BRANCO
```

## ▪ Caso não exista valor com esse nome

- Gerado erro de execução
  - Lançada **exceção** `IllegalArgumentException`
- Exemplo

```
Ex: Cor.valueOf ("branco")
```

## ▪ Exemplo

```
public class Exemplo {  
  
    ...  
    public static enum Cor { AMARELO, AZUL, BRANCO, ENCARNADO }  
    ...  
    public void metodo() {  
        Scanner ler = new Scanner(System.in);  
        System.out.println("Escreva uma cor (AMARELO, AZUL, BRANCO, ENCARNADO):");  
        String cor = ler.next().toUpperCase();           // utilizador define cor  
        ...  
        for( Object obj: figuras ) {                       // figuras é contentor ArrayList  
            Figura f = (Figura) obj;  
            if ( f.getCor() == Cor.valueOf( cor ) )        // getCor devolve valor do tipo Cor  
                System.out.println( f );                  // imprime só figuras com cor  
        }  
    }  
}
```

## ▪ Interesse

- Obter **ordem** (posição) de um valor // ordem = valor da constante enumerada

## ▪ Declaração

- `public int ordinal()` // método de instância

## ▪ Retorna

- **Ordem** (posição) do **valor** sobre o qual é aplicado o método

```
public class Exemplo {  
    public static enum Mes { JANEIRO, FEVEREIRO, MARCO, ABRIL, MAIO, JUNHO, JULHO,  
                             AGOSTO, SETEMBRO, OUTUBRO, NOVEMBRO, DEZEMBRO }  
  
    public void metodo() {  
        Mes m1 = Mes.DEZEMBRO;           // posição 11  
        Mes m2 = Mes.ABRIL;              // posição 3  
        System.out.println("Faltam " + m1.ordinal () – m2.ordinal () + " meses para o Natal");  
        ...  
    }  
}
```

- **Exemplo:** meses do ano - caracterizados pelo nome e nº de dias // conjunto fixo enumerado

```
public enum Mes {  
    // Valores do tipo (Instâncias do tipo)  
    Janeiro(31), Fevereiro(28), Marco(31) { public String toString() { return "Março"; } }, Abril(30), Maio(31),  
    Junho(30), Julho(31), Agosto(31), Setembro(30), Outubro(31), Novembro(30), Dezembro(31);  
  
    // Variável de instância  
    private int numeroDeDias;  
  
    // Construtor do tipo (privado)  
    private Mes( int numeroDeDias ) { this.numeroDeDias = numeroDeDias; }  
  
    // Métodos de instância (aplicados aos valores do tipo. Ex: int d = Mes.Janeiro.numeroDeDias(2013); )  
    public int numeroDeDias(int ano) {  
        if( this.ordinal()==1 && Mes.anoBissexto(ano) ) return this.numeroDeDias+1;  
        return this.numeroDeDias;  
    }  
  
    public int numeroDeDiasDoAnoCorrente() {  
        Calendar c = Calendar.getInstance();  
        if( this.ordinal()==1 && Mes.anoBissexto( c.get( Calendar.YEAR ) ) ) return this.numeroDeDias+1;  
        return this.numeroDeDias;  
    }  
  
    public int ordem(){ return this.ordinal()+1; }  
  
    // Métodos de classe (aplicados ao tipo. Ex: Mes m = Mes.obterMes(1); )  
    public static Mes obterMes( int ordem ) { return ordem>0 && ordem<13 ? Mes.values()[ordem-1] : null; }  
  
    private static boolean anoBissexto( int ano ) { return ano % 4 == 0 && ano % 100 != 0 || ano % 400 == 0; }  
}
```