

Tipos Enumerados

Programação Orientada a Objetos — QXD0007



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2021



Leituras para esta aula

- **Seção 6.10** e **Seção 8.9** do livro Java Como Programar, Décima Edição, Disponível no link: <http://libgen.lc/ads.php?md5=728636A04ACA056038BB5F079403AC96>

Enumerações

- Algumas vezes precisamos de uma variável que possa receber somente um certo conjunto finito e enumerado de valores. Além disso, em tempo de compilação, gostaríamos também de checar se essa variável possui um desses valores.

Enumerações

- Algumas vezes precisamos de uma variável que possa receber somente um certo conjunto finito e enumerado de valores. Além disso, em tempo de compilação, gostaríamos também de checar se essa variável possui um desses valores.
- Exemplos:
 - `dayOfWeek`: SUNDAY, MONDAY, TUESDAY, ...
 - `month`: JAN, FEB, MAR, APR, MAY, ...
 - `gender`: MALE, FEMALE ...

Enumerações

- Algumas vezes precisamos de uma variável que possa receber somente um certo conjunto finito e enumerado de valores. Além disso, em tempo de compilação, gostaríamos também de checar se essa variável possui um desses valores.
- Exemplos:
 - `dayOfWeek`: SUNDAY, MONDAY, TUESDAY, ...
 - `month`: JAN, FEB, MAR, APR, MAY, ...
 - `gender`: MALE, FEMALE ...
- Os valores estão escritos em caixa alta porque eles são constantes.
- Como implementar essas constantes?

Enumerações

- Antes da versão 1.5, o Java representava enumerações como valores inteiros :
 - `public final int SPRING = 0;`
`public final int SUMMER = 1;`
`public final int FALL = 2;`
`public final int WINTER = 3;`

Enumerações

- Antes da versão 1.5, o Java representava enumerações como valores inteiros :
 - `public final int SPRING = 0;`
`public final int SUMMER = 1;`
`public final int FALL = 2;`
`public final int WINTER = 3;`
- Infelizmente, esse tipo de abordagem permite que essas constantes sejam usadas onde quer que um inteiro seja esperado:
 - `int now = WINTER;`
 - `int month = now;`

Enumerações

- Antes da versão 1.5, o Java representava enumerações como valores inteiros :
 - `public final int SPRING = 0;`
`public final int SUMMER = 1;`
`public final int FALL = 2;`
`public final int WINTER = 3;`
- Infelizmente, esse tipo de abordagem permite que essas constantes sejam usadas onde quer que um inteiro seja esperado:
 - `int now = WINTER;`
 - `int month = now;`
- Hoje, o jeito correto de construir enumerações em Java é usando um tipo enumerado:
 - `enum Season { SPRING, SUMMER, FALL, WINTER };`

Enumerações

```
1
2 enum Color { RED, GREEN, BLUE };
3
4 public class Teste {
5     public static void main(String[] args) {
6         Color c = Color.RED;
7         System.out.println(c);
8     }
9 }
```

Enumerações

```
1
2 enum Color { RED, GREEN, BLUE };
3
4 public class Teste {
5     public static void main(String[] args) {
6         Color c = Color.RED;
7         System.out.println(c);
8     }
9 }
```

- A primeira linha de um `enum` deve ser uma lista de constantes. Depois dela, podem vir variáveis, construtores e métodos.
- Segundo as convenções do Java, todas as constantes devem ser escritas em maiúsculo.

enums são classes

- Todo **enum** é internamente implementado como uma classe.
- O **enum Color** do exemplo anterior é internamente convertido para:

```
class Color {  
    public static final Color RED = new Color();  
    public static final Color BLUE = new Color();  
    public static final Color GREEN = new Color();  
}
```

- Toda constante enum representa um **objeto** do tipo enum.

enums são classes

```
1 class Color {
2     public static final Color RED = new Color();
3     public static final Color BLUE = new Color();
4     public static final Color GREEN = new Color();
5 }
6
7 public class Teste2 {
8     public static void main(String[] args) {
9         Color c = Color.RED;
10        Color d = Color.RED;
11        if(c == d) {
12            System.out.println("c e d apontam pro mesmo objeto");
13        }
14        if(c.equals(d)) {
15            System.out.println("c e d são iguais");
16        }
17        System.out.println(c.toString());
18    }
19 }
```

enums são classes

- `enum` é um tipo de classe.
- Variáveis do tipo enum são verificadas em tempo de compilação
- Cada constante de uma enumeração é uma instância da classe `enum`
- Enums são implicitamente `public`, `static` e `final`
- Podemos comparar enums com `equals` ou `==`
- Enums sobrescrevem o método `toString()` e fornecem o método `valueOf()`
 - ```
Season season = Season.WINTER;
System.out.println(season); //prints WINTER
season = Season.valueOf("SPRING"); // atribui o valor
Season.SPRING
```

# Exemplo

```
1 enum Season {
2 SPRING, SUMMER, FALL, WINTER;
3 }
4
5 public class SeasonsTest {
6 public static void main(String[] args) {
7 Season s1 = Season.FALL;
8 Season s2 = Season.valueOf("WINTER");
9 Season s3 = Season.valueOf("FALL");
10
11 System.out.printf("%s\n%s\n%s\n", s1, s2, s3);
12
13 if(s1 == s3)
14 System.out.println("s1 e s3 são iguais");
15 else
16 System.out.println("s1 e s3 são diferentes");
17 }
18 }
```

## enums – Características

- Enums fornecem checagem de tipos em tempo de compilação
  - `int` não fornece essa checagem: `season = 43;`
- Enums podem ser usados no comando `switch`
- Enums são robustos. Se vocês adicionar, remover ou reordenar as constantes, você deve recompilar e tudo estará OK novamente.
- Como enums são objetos, podemos colocá-los em `collections`
- Como enums são classes, eles podem ter atributos, construtores e métodos.

## enums – Construtores

- Exceto pelos construtores, um `enum` é como uma classe qualquer.
- Cada nome listado dentro de um `enum` é, de fato, uma chamada para um construtor.
- Exemplo 1: `enum Season {WINTER, SPRING, SUMMER, FALL}`  
Essa linha constrói quatro objetos usando o construtor default.
- Os construtores de um `enum` só estão disponíveis **dentro do enum**, pois uma enumeração, uma vez definida, está completa e deve ser imutável.



## enums – Construtores

```
1 enum Coin {
2 PENNY(1), NICKEL(5), DIME(10), QUARTER(25);
3
4 private int value; // valor da moeda (em centavos)
5
6 // construtor do enum
7 Coin(int v) {
8 this.value = v;
9 }
10
11 public int getValue() { // getter
12 return value;
13 }
14
15 public void setValue(int v) { // setter
16 this.value = v;
17 }
18 }
19
20 public class CoinTest {
21 public static void main(String[] args) {
22 Coin c = Coin.NICKEL;
23 System.out.println(c + " vale " + c.getValue() + "
24 centavos");
```

# enums – Construtores

- Analisar o código `Semana.java` e `SemanaTest.java`

## enums – Métodos herdados

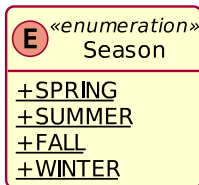
Todo `enum` implicitamente tem os seguintes métodos:

- `String toString()`: retorna o nome da constante enum, como contida na declaração.
- `boolean equals(Object obj)`: retorna true se o objeto especificado é igual a esta constante enum.
- `int ordinal()`: retorna o índice da constante enum.
- `int compareTo(Object obj)`: compara este enum com o objeto especificado com relação à ordem. Retorna um número inteiro negativo, zero, ou positivo de acordo se este objeto enum é menor que, igual ou maior que o objeto enum especificado.
- `static enum-type[] values()`: retorna um array com os objetos da enumeração.
- `static enum-type valueOf(String s)`: retorna o objeto enumerado cujo nome é *s*.

# Exemplo

- Analisar o arquivo [ExemploTeste.java](#)

# Diagrama de classes UML



- Uma **classe de enumeração** lista todos os valores válidos que um tipo de dados pode assumir e, embora não costume possuir associações, é geralmente colocada próxima das classes que utilizam o tipo de dados cujos literais são por ela enumerados.

FIM

