UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

AULA 11

LINGUAGEM DE PROGRAMAÇÃO 2
JAVA

PROF. JANIHERYSON FELIPE

CONTEÚDO DESSA AULA

- ENTENDER O CONCEITO DE ENUM E COMO UTILIZA-LAS;
- ENTENDER O QUE SÃO WRAPPERS E ONDE APLICA-LOS;
- ENTENDER O CONCEITO DE VARARGS;
- DISCUSSÕES E DÚVIDAS GERAIS.

ENUMERAÇÕES

- Enums são tipos de dados especiais que permitem definir um **conjunto fixo de constantes nomeadas**. Eles são úteis quando você tem um conjunto predefinido de valores que um variável pode ter. Por exemplo, se você estiver lidando com os **dias da semana**, **cores ou estados** de um objeto, você pode usar enums para representar esses valores de forma mais legível e segura.

ENUMERAÇÕES

```java
// Definindo uma enumeração para os dias da semana
public enum DiaDaSemana {
    SEGUNDA, TERÇA, QUARTA, QUINTA, SEXTA, SÁBADO, DOMINGO
}

    // Usando a enumeração
    public class Main {
        Run | Debug
        public static void main(String[] args) {
            DiaDaSemana dia = DiaDaSemana.SEGUNDA;
            System.out.println("Hoje é " + dia);
        }
    }
```

ENUMERAÇÕES

- A enumerações permitem que sejam utilizadas **construtures** e metodos **internos**.
- Não é necessario usar a palavra **new** para chamar o constutor de uma enum.
- Não usamos modificadores de acesso em construtores de enuns
- Isso permite que uma enum assuma dois valores possiveis

```java
public enum DiaDaSemana {
    SEGUNDA(dia:1),
    TERÇA(dia:2),
    QUARTA(dia:3),
    QUINTA(dia:4),
    SEXTA(dia:5),
    SÁBADO(dia:6),
    DOMINGO(dia:7);

    private final int dia;

    DiaDaSemana(int dia){
        this.dia = dia;
    }

    public int getDia() {
        return dia;
    }
}
```

ENUMERAÇÕES

```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        DiaDaSemana dia = DiaDaSemana.SEGUNDA;
        System.out.println("Hoje é " + dia.getDia());
    }
}
```

Podemos recuperar apenas o número relacionado ao dia

ENUMERAÇÕES

```
public class Main {
    Run | Debug
    public static void main(String[] args) {

        for(DiaDaSemana dia : DiaDaSemana.values()){
            System.out.println(dia);
        }
```

É possivel retornar um array com todos os dias da semana atraves do método values();

CLASSES WRAPPERS: CLASSES DE TIPOS PRIMITIVOS

```
short num1 = 1;
byte num2 = 2;
int num3 = 3;
long num4 = 4;
float num5 = 4.5f;
double num6 = 5.55555;
boolean num7 = false;
char c = 'a';
```

- Algumas linguagem são 100% orientadas a objetos. ex: Ruby;
- Esse não é o caso da linguagem Java, que possue os tipos primitivos;

CLASSES WRAPPERS: CLASSES DE TIPOS PRIMITIVOS

- Em Java, um "wrapper" é um tipo de classe que **encapsula** (ou "**embrulha**") um tipo primitivo de dados em um objeto. Isso é útil porque, em muitos casos, você precisa tratar os tipos primitivos como **objetos**. Por exemplo, ao trabalhar com coleções (como **Listas ou Conjuntos)**, você geralmente precisa de objetos em vez de tipos primitivos.

CLASSES WRAPPERS: CLASSES DE TIPOS PRIMITIVOS

- Integer: Para encapsular valores inteiros.
- Double: Para encapsular valores de ponto flutuante duplo.
- Boolean: Para encapsular valores booleanos.
- Character: Para encapsular valores de caracteres.
- Byte: Para encapsular valores de bytes.
- Short: Para encapsular valores de short.
- Long: Para encapsular valores longos.
- Float: Para encapsular valores de ponto flutuante.

AUTOBOXING E AUTOUNBOXING

```
//autoboxing
Short num7 = 1;
Byte num8 = 10;
Integer num9 = 100;
Long num10 = 100l; //new Long(10000l);
Float num11 = 3.5f; //new Float(3.5f);
Double num12 = 2.55555;
Boolean flag2 = true;
Character b = 'b';
```

```
//auto un-boxing
int num13 = num9; //num9.intValue();
```

Pode ser feito diretamente oa conversão entre classes e primitivos

VANTAGENS E DESVANTAGENS

Característica	Tipos Primitivos	Wrappers (Tipos de Referência)
Desempenho	Melhor (menor overhead)	Menor (custo de objetos, autoboxing/unboxing)
Uso de Memória	Menos memória	Mais memória (porque são objetos)
Valor Nulo	Não pode ser `null`	Pode ser `null`
Métodos	Nenhum método associado	Métodos úteis (e.g., `toString()`, `compareTo()`)
Coleções e Genéricos	Não pode ser usado diretamente (apenas em arrays)	Pode ser usado com generics e coleções
Autoboxing/Unboxing	Não há autoboxing/unboxing	Autoboxing/unboxing pode gerar overhead
Flexibilidade	Menos flexível em alguns casos	Mais flexível, especialmente em APIs que exigem objetos

VARARGS

```java
public static void main(String[] args) {
    int resultado = soma(...valores:5, 54, 87, 7);
    System.out.println(resultado);
}

public static int soma(Integer... valores){
    int valor = 0;
    for(int i = 0; i < valores.length; i++){
        valor += valores[i];
    }
    return valor;
}
```

Permite passar uma quantidade indeterminada de valores para uma função. substitui o uso de vetores;