

So28 – Estrutura de Dados

Professor MSc. Eng. Márcio Lemos

E-mail: marcio.lemos@senairs.org.br

<http://lattes.cnpq.br/4769158065464009>

Unidade Curricular

So28 – Estrutura de Dados

Carga Horária: 70 horas-aula

AULA 2

Enumerações; definições, aplicações e exemplos de encapsulamento, coesão e acoplamento. Recursão direta e indireta: Conceitos e aplicações, Desenvolvimento de algoritmos recursivos.

Exercícios de fixação.









Tipo enum em Java (enumerações)

- É um tipo especial que serve para especificar de forma literal um conjunto de constantes relacionadas

Tipo enum em Java (enumerações)

- Palavra chave em Java: **enum**

Tipo enum em Java (enumerações)

- **Vantagem:** melhor semântica, código mais legível e auxiliado pelo compilador
- **Referência:** <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>



Tipo enum em Java (enumerações)

- **Vantagem:** melhor semântica, código mais legível e auxiliado pelo compilador
- **Referência:** <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

Tipo enum em Java (enumerações)

```
package entities.enums;  
  
public enum OrderStatus {  
    AGUARDANDO_PAGAMENTO,  
    PROCESSANDO,  
    ENVIADO,  
    ENTREGUE;  
}
```


Tipo enum em Java (enumerações)











```
package entities;  
  
import java.util.Date;  
import entities.enums.OrderStatus;  
  
public class Order {  
    private Integer id;  
    private Date moment;  
    private OrderStatus status;  
    (to be continued...)
```

Tipo enum em Java (enumerações)

- Conversão de String para **enum**

OrderStatus os1 = OrderStatus.ENTREGUE;

OrderStatus os2 = OrderStatus.valueOf("ENTREGUE");

- ▼  Estrutura_Dados_Senai_Marcio2
 - >  JRE System Library [jre]
 - ▼  src
 - ▼  application
 - >  Program.java
 - ▼  entities
 - >  Order.java
 - ▼  entities.enums
 - ▼  OrderStatus.java
 - >  OrderStatus

```
Program.java ×
1 package application;
2
3 import java.util.Date;
4 import entities.Order;
5 import entities.enums.OrderStatus;
6 public class Program {
7     public static void main(String[] args) {
8
9         Order order = new Order(1080, new Date(), OrderStatus.PENDING_PAYMENT);
10        System.out.println(order);
11
12        OrderStatus os1 = OrderStatus.DELIVERED;
13        OrderStatus os2 = OrderStatus.valueOf("DELIVERED");
14
15        System.out.println(os1);
16        System.out.println(os2);
17    }
18 }
19 |
```











```
Program.java OrderStatus.java X
1 package entities.enums;
2
3 public enum OrderStatus {
4
5     PENDING_PAYMENT, //AGUARDANDO_PAGAMENTO
6     PROCESSING,       //PROCESSANDO
7     SHIPPED,           //ENVIADO
8     DELIVERED;         //ENTREGUE
9 }
10
```



```
Program.java OrderStatus.java Order.java X
1 package entities;
2
3 import java.util.Date;
4 import entities.enums.OrderStatus;
5 public class Order {
6
7     private Integer id;
8     private Date moment;
9     private OrderStatus status;
10
11     public Order() {
12         //Criando construtor vazio
13     }
14
15     public Order(Integer id, Date moment, OrderStatus status) {
16         this.id = id;
17         this.moment = moment;
18         this.status = status;
19     }
20
```

```
Program.java  OrderStatus.java  Order.java X
21 public Integer getId() {
22     return id;
23 }
24 public void setId(Integer id) {
25     this.id = id;
26 }
27
28 public Date getMoment() {
29     return moment;
30 }
31 public void setMoment(Date moment) {
32     this.moment = moment;
33 }
34
35 public OrderStatus getStatus() {
36     return status;
37 }
38 public void setStatus(OrderStatus status) {
39     this.status = status;
40 }
41
```

```
42 • @Override
43 public String toString() {
44     return "Order [id=" + id + ", moment=" + moment + ", "
45         + "status=" + status + "]\n";
46 }
47 }
48
```

- ▼  Estrutura_Dados_Senai_Marcio2
 - >  JRE System Library [jre]
 - ▼  src
 - ▼  application
 - >  Program.java
 - ▼  entities
 - >  Order.java
 - ▼  entities.enums
 - ▼  OrderStatus.java
 - >  OrderStatus

POO Classes

- Em um **sistema orientado a objetos**, de modo geral "tudo" é objeto.

POO Classes

- **Há várias categorias de classes:**
 - **Views** “Telas”
 - **Controllers** “Telas e Sistema”
 - **Entities** “Do negócio: Produto, Cliente, Pedido, etc.”
 - **Services**
 - **Repositories** “OBJ: Acesso dados do Banco de Dados”

Encapsulamento

- O **Encapsulamento** vem de encapsular, que em programação orientada a objetos significa **separar o programa em partes**, o mais isolado possível.

Encapsulamento

- A idéia é tornar o software mais flexível, **fácil de modificar** e de criar novas implementações.

Encapsulamento

- O Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe.

Encapsulamento

- É uma forma eficiente de **proteger os dados manipulados dentro da classe**, além de determinar onde esta classe poderá ser manipulada.



Encapsulamento

- Usamos o nível de acesso mais restritivo, **private**, que faça sentido para um membro particular.

Encapsulamento

- O encapsulamento que é dividido em dois níveis:
 - **Nível de classe:** Quando determinamos o acesso de uma classe inteira que pode ser public ou Package-Private (padrão);

Encapsulamento

- O encapsulamento que é dividido em dois níveis:
 - **Nível de membro:** Quando determinamos o acesso de atributos ou métodos de uma classe que podem ser public, private, protected ou Package-Private (padrão).

Encapsulamento


- Para se ter acesso a algum atributo ou método que esteja encapsulado utiliza-se o conceito de **get** e **set**.


Encapsulamento

- Por definição, com **SET** é feita uma atribuição a algum atributo, ou seja, define, diz o valor que algum atributo deve ter.
- E com **GET** é possível recuperar esse valor.

```
1 private String atributo1 = new String();
2 private String atributo2 = new String();
3 public String getAtributo1(){
4     return this.atributo1;
5 }
6 public String getAtributo2(){
7     return this.atributo2;
8 }
```




▼  Estrutura_Dados_Senai_Marcio2_1


>  JRE System Library [jre]


▼  src


▼  (default package)

>  Aluno.java

>  Estudos.java

>  Pessoa.java

>  Pessoa.java

>  Estudos.java

Aluno.java X

```
1 |  
2 public class Aluno extends Pessoa{  
3     public String matricula;  
4 }  
5
```

```
Aluno.java Pessoa.java X
1
2 public class Pessoa {
3     protected String nome;
4     protected String sobrenome;
5     protected String dataNasc;
6     protected String rg;
7     protected String telefones;
8     protected int idade;
9
10    public String getNome() {
11        return nome;
12    }
13    public void setNome(String nome) {
14        this.nome = nome;
15    }
16    public String getSobrenome() {
17        return sobrenome;
18    }
19    public void setSobrenome(String sobrenome) {
20        this.sobrenome = sobrenome;
21    }
```

```
Aluno.java Pessoa.java X
22 public String getDataNasc() {
23     return dataNasc;
24 }
25 public void setDataNasc(String dataNasc) {
26     this.dataNasc = dataNasc;
27 }
28 public String getRg() {
29     return rg;
30 }
31 public void setRg(String rg) {
32     this.rg = rg;
33 }
34 public String getTelefones() {
35     return telefones;
36 }
37 public void setTelefones(String telefones) {
38     this.telefones = telefones;
39 }
```

```
40 public int getIdade() {  
41     return idade;  
42 }  
43 public void setIdade(int idade) {  
44     this.idade = idade;  
45 }  
46 }  
47
```

```
Aluno.java Pessoa.java Estudos.java X
1 |
2 public class Estudos extends Aluno{
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         Aluno aluno = new Aluno();
8         aluno.nome = "Aluno Esforçado";
9         aluno.setIdade(24);
10        aluno.matricula = "825433551";
11
12        System.out.println("Nome: " + aluno.nome + "\n" +
13                            "Idade: " + aluno.getIdade() + "\n" +
14                            "Matrícula: " + aluno.matricula);
15    }
16 }
17
```



```
Console X
<terminated> Estudos [Java Application] C:\Users\samsu
Nome: Aluno Esforçado
Idade: 24
Matrícula: 825433551
```

Matrícula: 825433551

Coesão e Acoplamento

- São princípios de engenharia de software muito utilizados.

Coesão e Acoplamento

- Coesão está, na verdade, ligado ao **princípio da responsabilidade única**, que foi introduzido por Robert C. Martin no início dos anos 2000.

Coesão e Acoplamento

- Diz que **uma classe** deve ter apenas uma única **responsabilidade** e realizá-la de maneira satisfatória, ou seja;

Coesão e Acoplamento

- Uma classe não deve assumir responsabilidades **que não são suas**.

Coesão e Acoplamento

- Uma vez sendo ignorado este princípio, passamos a ter problemas, como dificuldades de manutenção e de reuso.

Coesão e Acoplamento

```
1 public class Programa
2 {
3     public void ExibirFormulario() {
4         //implementação
5     }
6
7     public void ObterProduto() {
8         //implementação
9     }
10
11     public void gravarProdutoDB {
12         //implementação
13     }
14 }
```


Coesão e Acoplamento

- Como visto no exemplo acima, a **classe Programa** tem responsabilidades que não são suas, como obter um produto e gravá-lo no banco de dados.

Coesão e Acoplamento

- Então, dizemos que esta classe não está coesa, ou seja, ela tem responsabilidades demais, e o que é pior, responsabilidades que não são suas.

Coesão e Acoplamento

- Vemos no **exemplo ABAIXO**, uma clara separação de responsabilidades, o que **contribui para um design desacoplado e organizado**.

Coesão e Acoplamento

```
1 public class Programa
2 {
3     public void MostrarFormulario() {
4         //Implementação
5     }
6
7     public void BotaoGravarProduto( ) {
8         Produto.gravarProduto();
9     }
10
11 }
```

Coesão e Acoplamento

- O **formulário** não assume o papel de cadastrar o produto, **ele pede a quem tem a responsabilidade para que faça tal tarefa.**



Coesão e Acoplamento

- O **Acoplamento** significa o quanto uma classe depende da outra para funcionar.

Coesão e Acoplamento

- E quanto maior for esta dependência entre ambas, dizemos que estas classes (elas estão fortemente acopladas).

Coesão e Acoplamento

- **ATENÇÃO!!** O forte acoplamento também nos traz muitos problemas. Cuidado!!

Recursividade

- Recursividade (ou Recursão) é um conceito em ciência da computação que se refere à **capacidade de uma função** ou **procedimento chamar a si mesmo de forma direta ou indireta** durante a sua execução.

Recursividade

- Em outras palavras, é quando uma função é definida em termos dela mesma.

Recursividade

- A recursividade é frequentemente usada para resolver problemas que podem ser subdivididos em subproblemas do mesmo tipo.

Recursividade

- Essa abordagem é especialmente útil quando a estrutura do problema é recursiva por natureza.

Recursividade

- Alguns exemplos de algoritmos recursivos comuns incluem a busca em árvores, a ordenação rápida (quicksort) e o cálculo do fatorial.

Recursividade

- Uma função recursiva geralmente possui dois componentes principais:
- **Caso base:** É uma condição especial que determina quando a recursão deve parar. Quando a condição do caso base é atendida, a função recursiva não faz mais chamadas a si mesma e retorna um resultado específico.

Recursividade

- Uma função recursiva geralmente possui dois componentes principais:
- **Caso recursivo:** É o caso em que a função se chama novamente com um problema menor ou uma instância simplificada do problema original.

Recursividade

- A chamada recursiva permite que a função trabalhe em direção ao caso base, resolvendo os subproblemas até que o resultado final seja alcançado.

Recursividade

- Existem **duas categorias** gerais de recursividade: **Recursão Direta** e **Recursão Indireta**.

Recursividade

- **Recursão Direta**
- Em uma recursão direta, uma função $f(A)$ chama a si própria diretamente – $f(A)$ chama $f(A)$, em um passo recursivo único.

Recursividade

- **Recursão Indireta**
- Já a recursividade indireta consiste em uma função $f(A)$ que chama uma função diferente $f(B)$, que por sua vez chama $f(A)$ novamente..

Recursividade

- **Recursão Indireta**
- Ou seja, é um processo recursivo de dois passos, onde a função original chama outra função que por sua vez chama a função original novamente.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int imprime(int inicio, int limite){
5      if(inicio<limite){
6          printf("%d ", inicio);
7          imprime(inicio+1, limite);
8      }
9  }
10
11 int main(){
12     int inicio, limite, i;
13
14     inicio=10;
15     limite=200;
16     printf("\n\nImpressao ITERATIVA\n");
17     for(i=inicio;i<limite;i++){
18         printf("%d ", i);
19     }
```

```
20     printf("\n\nImpressao RECURSIVA\n");
21     imprime(inicio, limite);
22     //imprime(10,200);
23     printf("\n\n");
24     system("pause");
25     return 0;
26 }
27
```

Impressao ITERATIVA

```
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 8
7 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 14
8 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 17
7 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
```

Impressao RECURSIVA

```
10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 8
7 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 14
8 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 17
7 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
```

sh: 1: pause: not found

...Program finished with exit code 0

Press ENTER to exit console.

RECURSIVIDADE

Recursividade é o processo de repetição de vários processos de forma similar, repetição essa contida no próprio processo. Exemplos disso são as imagens repetidas que aparecem quando dois espelhos são apontados um para o outro e as bonecas matrioscas, ou *bonecas russas*.

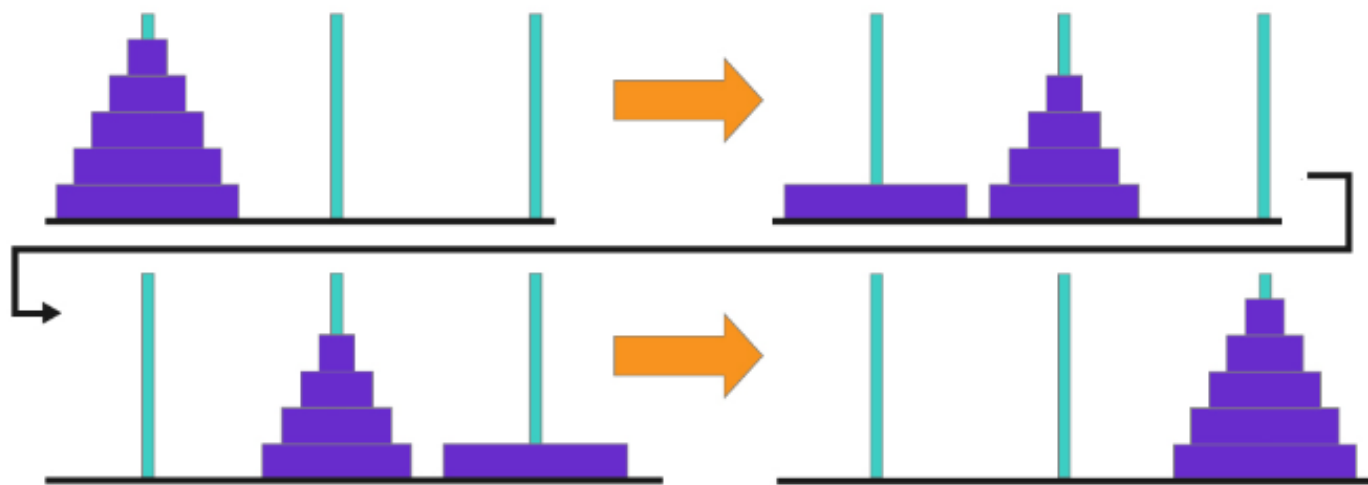
o outro e as bonecas matrioscas, ou bonecas russas.
repetidas que aparecem quando dois espelhos são apontados um para

A aplicação mais comum do conceito de recursividade se encontra nas disciplinas de Matemática e da Ciência da computação. *Recursividade* é nelas entendida como um método de definição de funções em que a função, ao ser definida dentro de seu corpo de instruções, chama a si própria. Como exemplos, tem-se o cálculo do fatorial e a torre de Hanói.

EXEMPLIFICANDO: TORRE DE HANÓI

Apesar de aparentemente complicado, esse problema tem uma solução recursiva simples:

- Para passar n peças de uma torre (A) para outra (C):
- Passar $n-1$ peças da torre inicial (A) para a torre livre (B).
- Mover a última peça para a torre final (C).
- Passar as $n-1$ peças da torre B para a torre final (C).



- Como fazer para calcular o fatorial do número 5? Ora, é $1*2*3*4*5$.
- E o fatorial do número 4? É $1*2*3*4$.
- E o fatorial do número 3? É $1*2*3$.
- E o fatorial do número 2? É $1*2$.
- E o fatorial do número 1? É 1.

Repare que o fatorial do número 3 é parte do cálculo do fatorial do número 4, ou seja, o fatorial de 4 é igual a 4 vezes o fatorial do 3, ou: $4! = 4 * 3!$

EXEMPLIFICANDO: FATORIAL

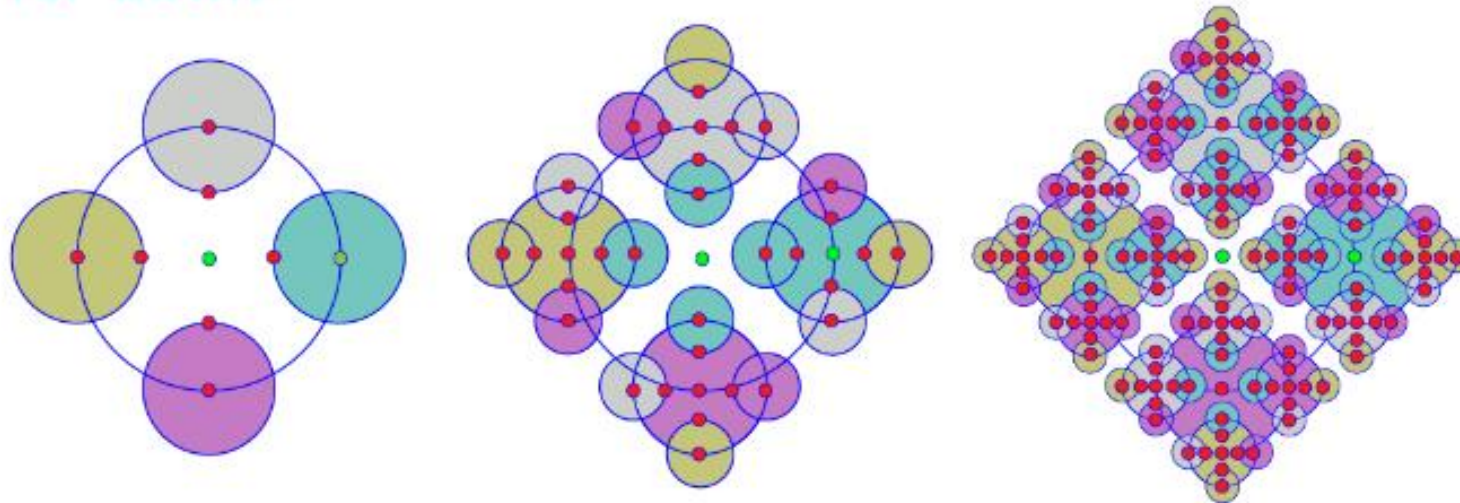
Fatorial (5)					
5*	Fatorial (5 - 1)				
	4*	Fatorial (4 - 1)			
		3*	Fatorial (3 - 1)		
			2*	Fatorial (2 - 1)	
				1*	Fatorial (1 - 1)
					1

DIVIDIR PARA CONQUISTAR

Um método comum de simplificação é dividir um problema em várias partes do mesmo tipo. Como técnica de programação de computador, chama-se *dividir* para conquistar.

Dividir para conquistar pode ser visto como uma análise *top-down* para resolução de problemas, onde estes são resolvidos pela resolução de várias instâncias dos mesmos, mas com menor dimensão. Exemplos de imagens envolvendo recursividade são os fractais geométricos, como no fractal que apelidamos de *tetra-círculo*, formado por circunferências com metade do raio original, construídas a partir dos pontos médios entre seus polos e seu centro, como na imagem a seguir.

TOP-DOWN



Representações dos 3 primeiros níveis de recorrência
para construir o fractal tetra-círculo.

para construir o fractal tetra-círculo.
representações dos 3 primeiros níveis de recorrência

O QUE É

MÉTODO ITERATIVO

A ideia é bem simples, um método iterativo é aquele que tem iterações, em outras palavras, é aquele onde você tem um loop e um contador. É comum usar a estrutura "para (for) ou enquanto (while)" para criar um método iterativo, embora existam outras maneiras.

MÉTODO RECURSIVO

É um método semelhante ao iterativo, porém ele não usa iterações, ele chama a mesma função diversas vezes e usa a memória para gravar seus resultados anteriores, isso pode estar meio confuso agora mas quando chegarmos ao código ficará mais simples.

ONDE PODE SER USADO

Um método iterativo pode ser usado para diversas coisas, porém, é mais fácil de perceber sua utilidade em operações matemáticas tais como somatórias e fatoriais (Note que existe um campo de estudos chamado métodos numéricos ou cálculo numérico, que se dedica ao estudo de métodos iterativos para resolver problemas tais como sistemas de equações, equações diferenciais, etc...).

Ele pode ser usado para substituir qualquer método iterativo, entretanto, sua implementação nem sempre é simples e muitas vezes o método recursivo será mais lento que o método iterativo.

COMPARANDO OS 2 MÉTODOS

Se ao implementar um método iterativo, fazemos com que o programa fique ligeiramente mais "pesado" por criar mais variáveis para armazenar dados, ao implementar um método recursivo fazemos um uso intenso da memória RAM do computador e este é um fator limitante para o método, uma vez que é possível que falte espaço na memória e neste caso provavelmente veremos nossa amiga tela azul (ou inimiga.. é uma tela divertida).

Normalmente métodos iterativos são mais rápidos que métodos recursivos, porém em alguns casos a implementação de um método recursivo é muito mais simples que a de um iterativo.

```
1 //Cálculo de fatorial com função recursiva
2 #include <stdio.h>
3 #include <conio.h>
4 //protótipo da função fatorial
5 double fatorial(int n);
6 int main(void)
7 {
8     int numero;
9     double f;
10    printf("Digite o numero que deseja calcular o fatorial: ");
11    scanf("%d",&numero);
12    //chamada da função fatorial
13    f = fatorial(numero);
14
15    printf("Fatorial de %d = %.0lf",numero,f);
16    getch();
17    return 0;
18 }
```

```
18 }
19 //Função recursiva que calcula o fatorial
20 //de um numero inteiro n
21 double fatorial(int n)
22 {
23     double vfat;
24
25     if ( n <= 1 )
26         //Caso base: fatorial de n <= 1 retorna 1
27         return (1);
28     else
29     {
30         //Chamada recursiva
31         vfat = n * fatorial(n - 1);
32         return (vfat);
33     }
34 }
```



Digite o numero que deseja calcular o fatorial: 5

Fatorial de 5 = 120

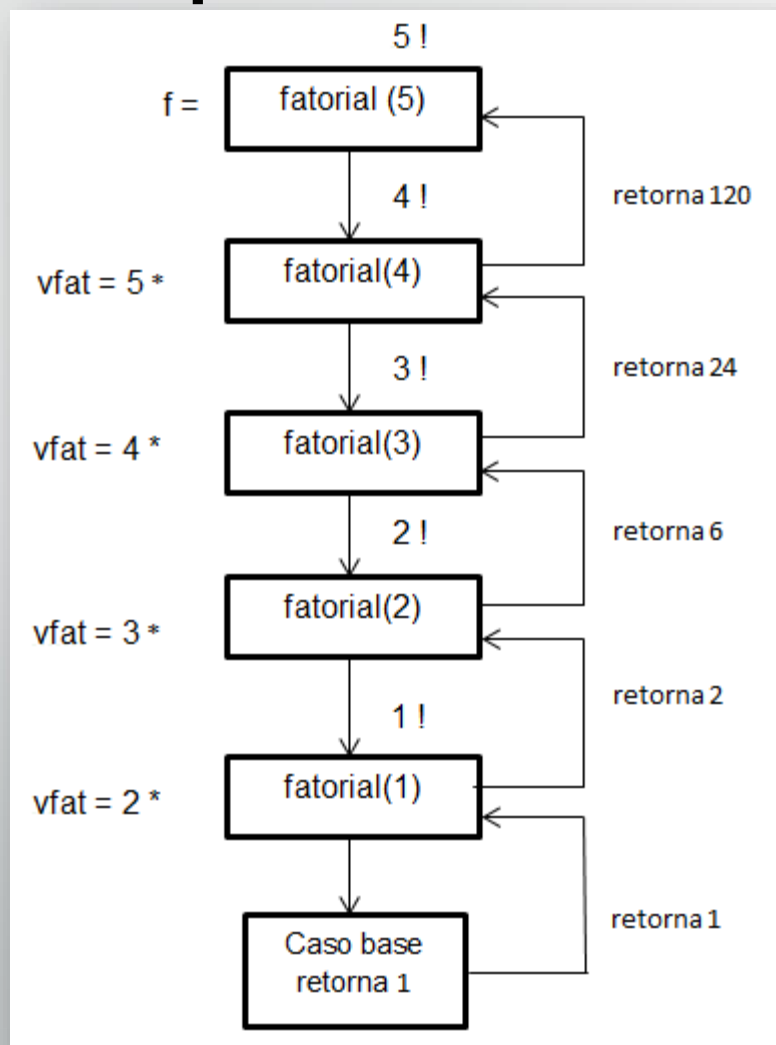
...Program finished with exit code 0

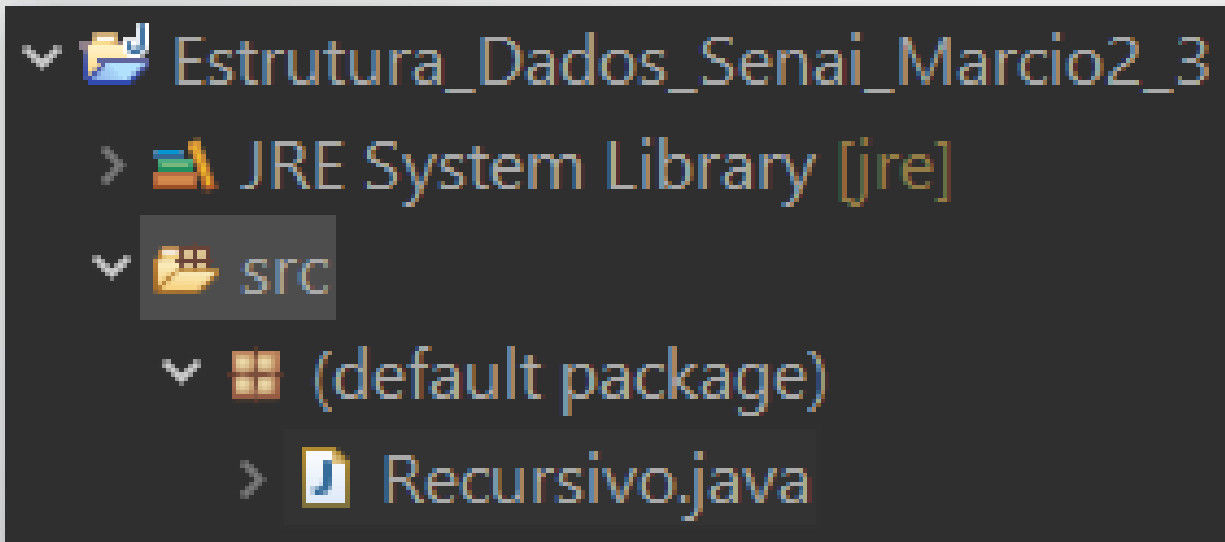
Press ENTER to exit console.

Fatorial

- Explicação do código
- No **programa acima**, se o número n for menor ou igual a 1 o valor 1 será retornado e a função encerrada, sem necessidade de chamadas recursivas.

- Veja a figura a seguir que representa as chamadas recursivas para o cálculo de $5!$











```
Recursivo.java ×
1 import java.util.Scanner;
2 public class Recursivo {
3     public static int fatorial(int n) {
4         if (n > 0) {
5             return n * Recursivo.fatorial(n - 1);
6         }
7         return 1;
8     }
9     public static void main(String [] args) {
10         System.out.print("Valor: ");
11         Scanner scan = new Scanner(System.in);
12         int valor = scan.nextInt();
13         System.out.println("O fatorial de " + valor + " é " +
14             Recursivo.fatorial(valor));
15         scan.close();
16     }
17 }
18
```

```
Console ×  
<terminated> Recursivo [Java Application] C:\Users\samsun  
Valor: 5  
O fatorial de 5 é 120
```




- ▼  Estrutura_Dados_Senai_Marcio2_4
 - >  JRE System Library [jre]
 - ▼  src
 - ▼  (default package)
 - >  DiasSemana.java
 - >  EnumsJava.java

```
DiasSemana.java × Enums.Java.java
1
2 public enum DiasSemana {
3     SEGUNDA,
4     TERÇA,
5     QUARTA,
6     QUINTA,
7     SEXTA,
8     SÁBADO,
9     DOMINGO
10 }
11
```

```
1
2 import java.util.Scanner;
3 public class EnumsJava {
4
5     public static void main(String[] args) {
6         Scanner entrada = new Scanner(System.in);
7         int dia = 0;
8         String diaSemana;
9         System.out.println("Entre com o código do dia da semana:");
10        dia = entrada.nextInt();
11        entrada.close();
12        switch (dia) {
13            case 1:
14                diaSemana = DiasSemana.DOMINGO.toString();
15                break;
16            case 2:
17                diaSemana = DiasSemana.SEGUNDA.toString();
18                break;
19            case 3:
20                diaSemana = DiasSemana.TERÇA.toString();
21                break;
```

```
22         case 4:
23             diaSemana = DiasSemana.QUARTA.toString();
24             break;
25         case 5:
26             diaSemana = DiasSemana.QUINTA.toString();
27             break;
28         case 6:
29             diaSemana = DiasSemana.SEXTA.toString();
30             break;
31         case 7:
32             diaSemana = DiasSemana.SÁBADO.toString();
33             break;
34         default:
35             diaSemana = "Dia inválido";
36     }
37     System.out.println("O dia da semana escolhido foi: " +
38     diaSemana);
39 }
40 }
41
```

Console X

<terminated> EnumsJava [Java Application] C:\Users\samsung\.p2\pool\plugins\org.eclipse.justj.openjdk.ho

Entre com o código do dia da semana:

1

O dia da semana escolhido foi: DOMINGO

Obrigado pela atenção

Sigo à disposição pelo e-mail:

marcio.lemos@senairs.org.br

