

# Modularização do código

PROFA. ANDRÉA APARECIDA KONZEN

FUNDAMENTOS DE PROGRAMAÇÃO

ESCOLA POLITÉCNICA - PUCRS

# Mas o que é modularização do código?

A programação modular consiste em **divisão de componentes ou subsistemas**.

Como se fossem contratos claros e dependências bem estabelecidas de acordo com a necessidade de cada aplicação.

# Mas o que é modularização do código?

Mecanismo que permite que um sistema de software seja dividido em partes que interagem entre si.

O código desenvolvido é dividido em módulos independentes, que podem ser utilizados por qualquer objeto e a qualquer tempo.



# MÉTODOS

Quando definimos um objeto num programa orientado a objetos, implementamos todo o comportamento desse objeto em **um ou mais métodos**.

Um método em Java é equivalente a uma função, subrotina ou procedimento em outras linguagens de programação.

Todos os métodos em Java devem sempre ser definidos dentro de uma classe.



# MÉTODOS

Os **métodos em Java** permitem que o código seja organizado em **blocos reutilizáveis** e simplificam a lógica do programa, tornando-o mais fácil de entender e manter.

# MÉTODOS

```
public static void main(String[] args)
```

Este é o método principal em Java, é o ponto de entrada para qualquer programa Java. Ele é responsável por iniciar a execução do programa e chamar outros métodos conforme necessário.

The diagram shows the Java method signature `public static void main(String args[])` in green text. Annotations with arrows point to specific parts of the signature: 'Access Specifier' points to 'public', 'Keyword' points to 'static', 'Return type' points to 'void', 'Method name' points to 'main', and 'Array of string type' points to 'String args[]'.

```
public static void main(String args[])
```

Annotations:

- Access Specifier (points to `public`)
- Keyword (points to `static`)
- Return type (points to `void`)
- Method name (points to `main`)
- Array of string type (points to `String args[]`)



# MÉTODOS

**Void:** é uma palavra-chave usada para especificar que um método não retorna nada. Como o método `main()` não retorna nada, seu tipo de retorno é nulo . Assim que o método `main()` termina, o programa java termina também. Portanto, não faz sentido retornar do método `main()` , pois a JVM não pode fazer nada com o valor de retorno dele.

**main:** É o nome do método principal Java. É o identificador que a JVM procura como ponto de partida do programa java. Não é uma palavra-chave.

**String [] args:** armazena argumentos de linha de comando Java e é uma matriz do tipo classe `java.lang.String` . Aqui, o nome do array String é `args`, mas não é fixo e o usuário pode usar qualquer nome no lugar dele.



# MÉTODOS

**Público:** é um modificador de acesso , que especifica de onde e quem pode acessar o método. Tornar o método `main()` público o torna globalmente disponível. É tornado público para que a JVM possa invocá-lo de fora da classe, pois não está presente na classe atual..

**Estatic:** É uma *palavra-chave* que quando associada a um método, torna-se um método relacionado à classe. O método *main()* é estático para que a JVM possa invocá-lo sem instanciar a classe. Isso também economiza o desperdício desnecessário de memória que teria sido usado pelo objeto declarado apenas para chamar o método *main()* pela JVM.



# Classes

Classes em Java são estruturas fundamentais da linguagem que permitem definir objetos e suas propriedades.

Servem como modelos que podem ser usados para criar objetos, ou instâncias dessas classes, com um conjunto específico de propriedades.



# Classes

As classes são compostas de variáveis de instância e métodos. As variáveis de instância são propriedades ou dados que pertencem a cada objeto criado a partir da classe.

Os métodos são operações que podem ser realizadas em um objeto da classe.



# Classes

```
public class Carro {  
    String marca;  
    String modelo;  
    int ano;  
  
    public void acelerar() {  
        System.out.println("O carro está acelerando...");  
    }  
  
    public void frear() {  
        System.out.println("O carro está freando...");  
    }  
}
```

Neste exemplo, a classe Carro tem três variáveis de instância: marca, modelo e ano, que descrevem as propriedades de um carro.

Ela também tem dois métodos: acelerar e frear, que representam as operações que podem ser realizadas em um carro.



# Objetos

Para criar um objeto a partir da classe do exemplo, podemos usar a palavra-chave **new** seguida pelo **nome da classe** e um **conjunto de parênteses vazios**, indicando que nenhum argumento é passado para o construtor padrão:

```
Carro meuCarro = new Carro();
```



# Objetos

Este comando cria um novo objeto do tipo **Carro** e o armazena na variável **meuCarro**. Podemos então acessar as variáveis de instância e métodos deste objeto usando o **operador ponto**

```
meuCarro.marca = "Toyota";  
meuCarro.modelo = "Corolla";  
meuCarro.ano = 2021;
```



# Bibliotecas

Em Java, uma biblioteca é um conjunto de classes e métodos que fornecem funcionalidades específicas para serem usadas em um programa. O Java vem com muitas bibliotecas padrão, que podem ser usadas para realizar tarefas comuns, como entrada/saída de dados, manipulação de arquivos, operações matemáticas, etc.



# Bibliotecas

A importação de uma biblioteca em Java é feita usando a palavra-chave **import** seguida do nome da biblioteca ou classe que se deseja usar.

Por exemplo, para usar a classe Scanner, que é uma classe padrão do Java usada para ler dados de entrada do usuário, devemos primeiro importá-la para o nosso programa.



# Bibliotecas

```
import java.util.Scanner;
```

A palavra-chave **import** indica que estamos importando uma classe da biblioteca padrão do Java.

**java.util** é o nome da biblioteca que contém a classe Scanner, e **Scanner** é o nome da classe que estamos importando.



# Bibliotecas

Depois de importar a classe, podemos criar um objeto dela em nosso programa e usá-lo para ler dados de entrada do usuário.

## Exemplo:

```
import java.util.Scanner;

public class ExemploScanner {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Digite um número: ");
        int num = input.nextInt();
        System.out.println("Você digitou o número " + num);
    }
}
```



# Bibliotecas

No exemplo, foi criado um objeto **Scanner** chamado **input** e usamos o método **nextInt()** para ler um número inteiro digitado pelo usuário.

O método `nextInt()` espera que o usuário digite um número inteiro e retorna esse valor como um tipo inteiro. Em seguida, exibimos uma mensagem que inclui o número digitado pelo usuário.



# Bibliotecas

A importação de bibliotecas em Java é uma maneira fácil e poderosa de estender as funcionalidades do seu programa, aproveitando o código já existente em bibliotecas padrão ou em bibliotecas de terceiros.

Com a importação correta das bibliotecas, podemos aproveitar classes e métodos que são úteis em nossos programas e tornar nosso código mais eficiente e legível.



# Para lembrar

## Tipos de dados primitivos

### Números inteiros

Tipo	Tamanho	Valor
byte	8 bits	-128 a 127
short	16 bits	-32.768 a 32.767
int	32 bits	-2.147.483.648 a 2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807



# Para lembrar

## **Tipos de dados primitivos**

### Números em ponto flutuante

Tipo	Tamanho	Valor
float	32 bits	-3.40292347E+38 a +3.40292347E+38
double	64 bits	-1.79769313486231570E+308 a +1.79769313486231570E+308



# Para lembrar

## **Tipos de dados primitivos**

### Caracteres

Tipo  
char

Tamanho  
16 bits

Valor  
'\u0000' a '\uFFFF'



# Para lembrar

## Tipos de dados primitivos

### Caracteres

Códigos de escape de caráter	
Escape	Significado
\n	nova linha
\t	tabulação
\b	passo para trás
\r	retorno do carro
\\	barra invertida
\'	apóstrofe
\"	aspas



# Para lembrar

## **Tipos de dados primitivos**

### Booleanos

Tipo  
boolean

Tamanho  
1 bit

Valor  
true ou false



# Como trabalhamos até agora?

- tudo implementado no método main
- o método main como responsável pela implementação de funcionalidades do problema

# Como trabalhamos até agora?

```
1 import java.util.Scanner;
2 public class ExemploSemMetodo
3 {
4     public static void main(String args[]){
5         Scanner in = new Scanner(System.in);
6         int n,p,cont;
7         double fatN, fatNP, fatP;
8
9         System.out.print("Informe os valores de n e p: ");
10        n = in.nextInt();
11        p = in.nextInt();
12
13        fatN = 1;
14        for(cont=1; cont<=n; cont++){
15            fatN = fatN * cont;
16        }
17
18        fatP = 1;
19        for(cont=1; cont<=p; cont++){
20            fatP = fatP * cont;
21        }
22
23        fatNP=1;
24        for(cont=1;cont<=n-p; cont++){
25            fatNP = fatNP * cont;
26        }
27
28        System.out.println("Combinações: " + fatN/(fatP * fatNP));
29    }
30 }
```

Estão ligados?



# Qual nosso próximo passo?

- usar o método main para chamadas de outros métodos, tanto quanto possível.
- ser utilizado principalmente como um método de acesso a outros métodos da classe ou de outras classes.



# MÉTODOS

Agora que revisamos cada conceito do método principal, vamos às novidades...

Vamos criar **outros métodos** que sejam **responsáveis por funcionalidades específicas do problema a ser resolvido.**



# MÉTODOS

Como chamamos a novidade?

**Modularização de classes ou  
modularidade**





# MÉTODOS

São definidos dentro de uma classe e são compostos pelos seguintes elementos:

**Modificador de acesso:** indica a visibilidade do método  
(public, private, protected ou default)

**Tipo de retorno:** indica o tipo de valor que o método retorna  
(void, int, double, String, entre outros)



# MÉTODOS

**Nome do método:** um identificador único que deve seguir as convenções de **nomenclatura do Java**

**Parâmetros:** valores que o método espera receber para executar sua ação.

**Os parâmetros são opcionais e podem ter um ou mais tipos diferentes.**



# MÉTODOS

**Corpo do método:** o bloco de código que contém as instruções a serem executadas quando o método é chamado. Um método **pode ser chamado de dentro da mesma classe ou de outra classe**, dependendo de sua visibilidade e da instância da classe na qual ele foi definido.

*Para chamar um método de outra classe, é necessário criar um objeto dessa classe e usar sua referência para chamar o método.*



# MÉTODOS

```
1 import java.util.Scanner;
2 public class ExemploComMetodo
3 {
4     public static void main(String args[]){
5         Scanner in = new Scanner(System.in);
6         int n,p;
7
8
9         System.out.print("Informe os valores de n e p: ");
10        n = in.nextInt();
11        p = in.nextInt();
12
13
14        System.out.println("Combinações: " + fatorial(n)/(fatorial(p)*fatorial(n-p)));
15    }
16
17    public static double fatorial(int valor){
18        double f=1;
19        int cont;
20
21        for(cont=1; cont<=valor; cont++){
22            f = f * cont;
23        }
24
25        return f;
26    }
27 }
```





# MÉTODOS

```
1 import java.util.Scanner;
2 public class ExemploComMetodo
3 {
4     public static void main(String args[]){
5         Scanner in = new Scanner(System.in);
6         int n,p;
7
8
9         System.out.print("Informe os valores de n e p: ");
10        n = in.nextInt();
11        p = in.nextInt();
12
13
14        System.out.println("Combinações: " + fatorial(n)/(fatorial(p)*fatorial(n-p)));
15    }
16
17    public static double fatorial(int valor){
18        double f=1;
19        int cont;
20
21        for(cont=1; cont<=valor; cont++){
22            f = f * cont;
23        }
24
25        return f;
26    }
27 }
```

Aqui estamos fazendo o uso do método

Este é o método



# MÉTODOS

```
1 import java.util.Scanner;
2 public class ExemploComMetodo
3 {
4     public static void main(String args[]){
5         Scanner in = new Scanner(System.in);
6         int n,p;
7
8
9         System.out.print("Informe os valores de n e p: ");
10        n = in.nextInt();
11        p = in.nextInt();
12
13
14        System.out.println("Combinações: " + fatorial(n)/(fatorial(p)*fatorial(n-p)));
15    }
16
17    public static double fatorial(int valor){
18        double f=1;
19        int cont;
20
21        for(cont=1; cont<=valor; cont++){
22            f = f * cont;
23        }
24
25        return f;
26    }
27 }
```

**Return:** finaliza a execução de uma função e especifica os valores que devem ser retornados para onde a função foi chamada

O programa chama a **função fatorial()** passando o **valor de n como argumento** para calcular o fatorial de n. Em seguida, a fórmula matemática é aplicada para calcular o número de combinações possíveis entre n e p.

**A passagem de valores como argumentos** permite que você envie informações de uma parte do seu programa para outra (código modular e reutilizável). Aqui estamos fazendo **passagem por valor**.

Na passagem por valor, **o valor da variável original é copiado e enviado para o método como um argumento**. Quando o método altera o valor da variável recebida, **a variável original não é afetada**. Ou seja, **uma cópia do valor original é passada** e, portanto, qualquer modificação feita no valor copiado não afeta o valor original.



# MÉTODOS (outros métodos básicos)

**public static int soma(int a, int b)**

recebe dois números inteiros como argumentos e retorna a soma dos dois números.

**public static String concatenar(String a, String b)**

recebe duas strings como argumentos e retorna uma nova string que é a concatenação das duas strings.



# MÉTODOS (outros métodos básicos)

**public static boolean ehPar(int numero)**

recebe um número inteiro como argumento e retorna true se o número é par e false se o número é ímpar.

**public static double media(double[] numeros)**

recebe um array de números do tipo double como argumento e retorna a média dos números.



# MÉTODOS (outros métodos básicos)

**public static void imprimir(String texto)**

recebe uma string como argumento e imprime a string no console.

**public static int fatorial(int n)**

recebe um número inteiro como argumento e retorna o fatorial desse número.



# MÉTODOS (outros métodos básicos)

**public static void ordenar(int[] numeros)**

recebe um array de números inteiros como argumento e o ordena em ordem crescente.

**public static boolean ehPrimo(int numero)**

recebe um número inteiro como argumento e retorna true se o número é primo e false se o número não é primo.



E aí, o que acharam?

