



Programação OO

4ª Aula

Prof. Douglas Oliveira

douglas.oliveira@prof.infnet.edu.br

- Nos exemplos e exercícios implementados até agora, sempre executamos métodos **a partir de objetos**. Exemplos:

```
r1.mover(10, 20);           // move o objeto r1
```

```
r2.redimensionar(2, 3);    // redimensiona o objeto r2
```

- Em alguns casos precisamos implementar alguns métodos que **independem dos objetos** de uma classe.
- Normalmente esses métodos implementam serviços que estão **associados à classe** e não a um objeto específico.

- Se pensarmos na implementação do método para calcular a raiz quadrada, qual seria o objeto sobre o qual aplicaríamos esse método ?
- Pense, nas demais funções matemáticas: seno, cosseno, tangente, exponenciação, valor absoluto, logaritmo, arredondamento, etc.
- Nesse caso, o que queremos na realidade é implementar um conjunto de funções matemáticas que **independem de um objeto específico**.
- Nesse caso, chamamos esses métodos de **métodos da classe**, que são implementados usando a palavra **static**.

- No Java existe uma classe bastante usada que implementa funções matemáticas chamada **Math**.
- **Todos** os métodos dessa classe são **estáticos**, isto é não dependem de nenhum objeto para serem executados.

```
public class Math {  
    public static double sqrt(double x) { ... }  
    public static long round(double x) { ... }  
    public static double pow(double x, double y) { ... }  
}
```

- A principal diferença entre os métodos estáticos e os métodos de instância está na sua **chamada**.
- Nos métodos estáticos **não precisamos do objeto** para chamar o método.
- Ao invés disso, usamos o próprio **nome da classe** para chamá-los.
- Exemplos:

```
double x = 12.2;
```

```
double y = Math.sqrt(x); //calcula a raiz quadrada de x
```

```
long    z = Math.round(y); //arredonda o valor de y
```

Método	Descrição
abs(x)	Retorna o valor absoluto de x. x pode ser do tipo int, long, float ou double.
double acos(double x)	Retorna o arco coseno do parâmetro x
double asin(double x)	Retorna o arco seno do parâmetro x
double atan(double x)	Retorna o arco tangente do parâmetro x
double ceil(double x)	Retorna o menor inteiro maior que x
double cos(double x)	Retorna o coseno de x
double exp(double x)	Retorna a constante de Euler e elevada a x
double floor(double x)	Retorna o maior inteiro menor que x
double log(double x)	Retorna o logaritmo natural de x
max(x, y)	Retorna o maior valor entre x e y. x e y podem ser do tipo int, long, float ou double.
min(x, y)	Retorna o menor valor entre x e y. x e y podem ser do tipo int, long, float ou double.
double pow(double x, double y)	Retorna x^y
double random()	Retorna um número aleatório maior ou igual a ZERO e menor que UM
long round(double x)	Retorna o inteiro mais próximo de x
double sin(double x)	Retorna o seno de x
double tan(double x)	Retorna a tangente de x
double sqrt(double x)	Retorna a raiz quadrada de x

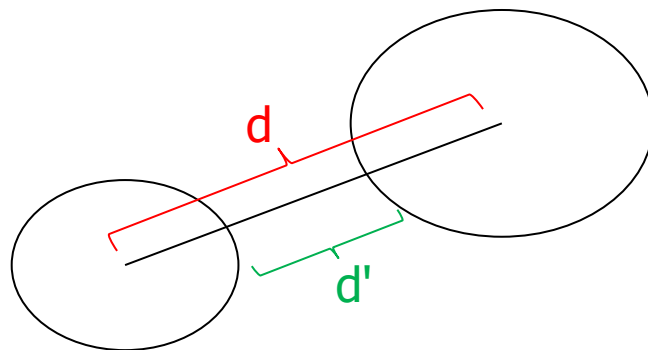
- Exercício 26: implemente na classe `Circulo` o método:

```
float distancia(Circulo outroCirculo)
```

que calcula a distância **d'** de um círculo a outro círculo.

Dica: a distância entre dois pontos é dada pela fórmula:

$$d = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$$



- Vimos anteriormente que métodos estáticos são métodos acessados **diretamente pela classe** e não por um objeto específico.
- Na classe Math, os métodos estáticos foram usados para criar uma biblioteca de métodos que implementam funções matemáticas.
- Entretanto, também é possível definir **atributos estáticos**.
- No caso de atributos estáticos só existe **UMA** cópia desse atributo na memória que é **COMPARTILHADA** por todos os objetos da classe.
- Assim como os métodos estáticos, os atributos estáticos são chamados **atributos da classe**, ou seja, podem ser acessados **diretamente a partir da classe**.

Exemplo:

```
public class Carro {  
    private int anoFabricacao;  
    static private int qtdeCarros = 0;  
    public Carro(int ano) {  
        anoFabricacao = ano;  
        // Cada vez que eu crio um objeto Carro, eu incremento o contador  
        qtdeCarros ++;  
    }  
    public int getAnoFabricacao() {  
        return anoFabricacao;  
    }  
    public static int getQtdeCarros() {  
        return qtdeCarros;  
    }  
}
```

Independentemente de quantos objetos sejam criados, só existe **uma** área de memória para o atributo **qtdeCarros**.

Essa área de memória é **compartilhada** por todos os objetos.

Esse método é de instância e só pode ser chamado pelo **objeto**.

Esse método é **estático** e deve ser chamado pela **classe**.

□ Exemplo (continuação): o que acontece na memória nesse caso ?

```
public class Concessionaria {  
    public static void main(String args[]) {
```

```
        Carro c1 = new Carro(2008);  
        Carro c2 = new Carro(2011);  
        Carro c3 = new Carro(2012);  
        Carro c4 = new Carro(2010);
```

```
        System.out.println("Ano de c1 = " + c1.getAnoFabricacao());  
        System.out.println("Ano de c2 = " + c2.getAnoFabricacao());  
        System.out.println("Ano de c3 = " + c3.getAnoFabricacao());  
        System.out.println("Ano de c4 = " + c4.getAnoFabricacao());
```

```
        System.out.println("Qtde de carros = " + Carro.getQtdeCarros());
```

```
    }
```

```
}
```

anoFabricacao = 2008

qtdeCarros= 1

Memória compartilhada pelos objetos !
Qualquer alteração nesse atributo realizado por um objeto é "enxergado" por todos os objetos dessa classe.

□ Exemplo (continuação): o que acontece na memória nesse caso ?

```
public class Concessionaria {  
    public static void main(String args[]) {  
  
        Carro c1 = new Carro(2008);  
        Carro c2 = new Carro(2011);  
  
        Carro c3 = new Carro(2012);  
        Carro c4 = new Carro(2010);  
  
        System.out.println("Ano de c1 = " + c1.getAnoFabricacao());  
        System.out.println("Ano de c2 = " + c2.getAnoFabricacao());  
        System.out.println("Ano de c3 = " + c3.getAnoFabricacao());  
        System.out.println("Ano de c4 = " + c4.getAnoFabricacao());  
  
        System.out.println("Qtde de carros = " + Carro.getQtdeCarros());  
    }  
}
```

anoFabricacao = 2008

anoFabricacao = 2011

qtdeCarros= 2

Memória compartilhada
pelos objetos !
Qualquer alteração nesse
atributo realizado por um
objeto é "enxergado" por
todos os objetos dessa
classe.

□ Exemplo (continuação): o que acontece na memória nesse caso ?

```
public class Concessionaria {  
    public static void main(String args[]) {  
        Carro c1 = new Carro(2008);  
        Carro c2 = new Carro(2011);  
        Carro c3 = new Carro(2012);  
        Carro c4 = new Carro(2010);  
  
        System.out.println("Ano de c1 = " + c1.getAnoFabricacao());  
        System.out.println("Ano de c2 = " + c2.getAnoFabricacao());  
        System.out.println("Ano de c3 = " + c3.getAnoFabricacao());  
        System.out.println("Ano de c4 = " + c4.getAnoFabricacao());  
  
        System.out.println("Qtde de carros = " + Carro.getQtdeCarros());  
    }  
}
```

anoFabricacao = 2008

anoFabricacao = 2011

anoFabricacao = 2012

anoFabricacao = 2010

qtdeCarros = 4

Memória compartilhada
pelos objetos !
Qualquer alteração nesse
atributo realizado por um
objeto é "enxergado" por
todos os objetos dessa
classe.

□ Exemplo (continuação):

```
public class Concessionaria {  
    public static void main(String args[]) {  
        Carro c1 = new Carro(2008);  
        Carro c2 = new Carro(2011);  
        Carro c3 = new Carro(2012);  
        Carro c4 = new Carro(2010);
```

Será impresso:

```
Ano de c1 = 2008  
Ano de c2 = 2011  
Ano de c3 = 2012  
Ano de c4 = 2010  
Qtde de carros = 4
```

```
        System.out.println("Ano de c1 = " + c1.getAnoFabricacao());  
        System.out.println("Ano de c2 = " + c2.getAnoFabricacao());  
        System.out.println("Ano de c3 = " + c3.getAnoFabricacao());  
        System.out.println("Ano de c4 = " + c4.getAnoFabricacao());  
        System.out.println("Qtde de carros = " + Carro.getQtdeCarros());
```

```
    }
```

```
}
```

- Algumas diferenças importantes:

Métodos de Instância	Métodos Estáticos
Só podem ser chamados a partir de objetos da classe.	São chamados a partir da própria classe.
Podem acessar (ler ou alterar) atributos da instância ou atributos da classe (estáticos).	Só podem acessar (ler ou alterar) atributos da classe (estáticos).

- Podemos definir **atributos estáticos públicos** junto com a palavra **final** para criar **constantes** que pertencem a uma **classe**.
- Um exemplo é a constante **PI** que está definida na classe **Math**:

```
public class Math {  
    public static final double PI = 3.14159;  
}
```

- Assim, para usar **PI** usamos a classe **Math**:

```
double raio = 7.5;  
double area = Math.PI * Math.pow(raio, 2);    //  $\pi R^2$ 
```

- Semelhante ao C/C++, o Java também dá suporte à vetores e matrizes multidimensionais.
- Entretanto, existem algumas diferenças importantes. A primeira delas é que, em Java, **vetores são objetos**.
- Para declarar um vetor, usamos o operador `[]` imediatamente após o tipo desejado:

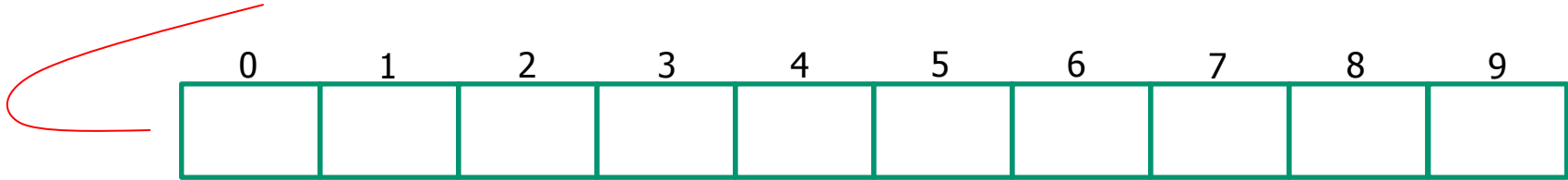
```
int[] v;
```

```
float[] notas;
```

- Outra diferença importante é que, ao declarar um vetor, **NÃO** definimos o seu tamanho.
- Isso significa dizer que, ao declarar um vetor, o Java **NÃO** aloca espaço na memória para o vetor.

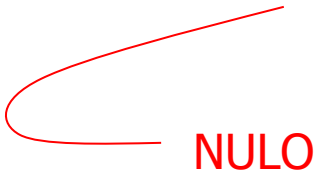
□ Em C/C++:

```
float nota[10];
```



□ Em Java:

```
float[] nota;
```



A red curved line points from the array declaration 'float[] nota;' to the word 'NULO' written in red capital letters.

- Como vetor é um **objeto**, precisamos criá-lo com o operador **new**. Nesse momento definimos o seu **tamanho**.

```
float[] nota;
```

```
nota = new float[10];
```

Aqui definimos o tamanho do vetor !

0	1	2	3	4	5	6	7	8	9
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

O índice inicial é sempre ZERO.

Os elementos do vetor são inicializados automaticamente da mesma forma que os atributos das classes.

O índice final é sempre o tamanho do vetor - 1.

- Diferentemente do C/C++, o tamanho do vetor pode ser definido usando uma **variável** ou uma **expressão**.

```
public class TesteVetor {  
    public static void main(String[] args) {  
        Scanner t = Scanner(System.in);  
        int tam;  
        float[] v;  
        System.out.print("Qual o tamanho do vetor ? ");  
        tam = t.nextInt();  
        v = new float[tam];  
    }  
}
```

Cria o vetor **v** usando como tamanho a variável **tam**.

- ❑ Atenção: uma vez definido o tamanho do vetor com o operador new **não podemos mais alterar esse tamanho.**

```
public class TesteVetor {  
    public static void main(String[] args) {  
        Scanner t = Scanner(System.in);  
        int tam;  
        float[] v;  
        System.out.print("Qual o tamanho do vetor ? ");  
        tam = t.nextInt();  
        v = new float[tam];  
        v[0] = 2;  
        v = new float[2 * tam];  
    }  
}
```

Nesse caso **NÃO** estamos aumentando o vetor v.

No primeiro **new** criamos um vetor com tamanho tam.

No segundo **new** criamos **outro** vetor com tamanho 2*tam.

- Cuidado: se você acessar uma área fora do intervalo do vetor, ocorrerá uma exceção (*ArrayIndexOutOfBoundsException*) e a execução do programa será interrompida.

```
float[] nota;  
nota = new float[10];  
nota[-1] = 7.5;  
nota[10] = 8.0;
```

Erro de execução:
ArrayIndexOutOfBoundsException

- Assim como fazemos com variáveis comuns, também podemos inicializar vetores.

```
tipo[] nome = { lista de valores };
```

- onde:

lista de valores é uma lista cujos elementos são separados por vírgula.

- Exemplos:

```
float[] nota = { 5.5, 6.5, 7.0, 9.0, 8.0 };
```

```
int[] idade = { 30, 22, 45, 12, 65, 48, 72 };
```

Cria automaticamente um vetor de 5 posições

Cria automaticamente um vetor de 7 posições

- ❑ Para percorrer individualmente cada elemento do vetor usamos o comando **for**.
- ❑ O atributo **length** do vetor pode ser usado para recuperar o tamanho do vetor.

```
public class Vetor {  
    public static void main(String[] args) {  
        int[] v = { 2, 4, 6, 8, 10 };  
  
        for (int i = 0; i < v.length; i++)  
            System.out.println(v[i]);  
    }  
}
```

Recupera o tamanho do vetor,
que é 5 nesse caso.

- Existe uma outra sintaxe do comando **for** para **percorrer vetores**. Essa forma é usada apenas para ler os dados do vetor, mas **não serve para alterar o vetor**.
- Usando essa nova sintaxe, o programa anterior poderia ser reescrito como:

```
public class Vetor {  
    public static void main(String[] args) {  
        int[] v = { 2, 4, 6, 8, 10 };  
        for (int n : v)  
            System.out.println(n);  
    }  
}
```


- Usando essa nova sintaxe, o programa anterior poderia ser reescrito como:

```
public class Vetor {  
    public static void main(String[] args) {  
        int[] v = { 2, 4, 6, 8, 10 };  

```

Tipo armazenado
no vetor

Variável que
percorre o vetor

Vetor

```
for (int n : v)
```

```
    System.out.println(n);  
}
```

```
}
```

A cada iteração, a variável **n** recebe os valores armazenados no vetor **v**, do primeiro até o último.

- Exercício 23: crie um programa para ler inicialmente o número de alunos de uma turma. Em seguida, leia as notas dos alunos dessa turma. Ao final, imprima a média da turma e as notas acima da média.

- Exercício 24: crie um programa para ler um número **n** e depois ler um vetor **v** com **n** números inteiros positivos. Ao final, ler mais um número **k** e informar e que posições do vetor **v** aparece o número **k**. Caso **k** não exista no vetor **v** apresente a mensagem "Número não encontrado".

- Exercício 25: cria a classe `ConjuntoDeReais` que deve armazenar um **conjunto de floats** usando um **vetor**. Implemente os métodos:

```
public ConjuntoDeReais(int qtdMaximaNumeros)
public boolean pertence(float n)
public boolean adicionar(float n)
public void imprimir()
public boolean remover(float n)
public int quantidade()
```

- ❑ Java não possui um tipo primitivo string como em algumas outras linguagens (VB, Delphi).
- ❑ A biblioteca de classes do Java possui uma classe pré-definida chamada **String**.
- ❑ Logo, em Java, strings não são tipos primitivos e sim objetos.
- ❑ A declaração de um objeto String segue o mesmo padrão de declaração das variáveis de tipos básicos:

```
String mensagem;
```

```
String nome = "Joao da Silva";
```

```
String saudacao = "Olá " + nome;
```

```
String vazia = "";
```

A concatenação de strings
é feita com o operador +

- ❑ Repare que não precisamos do operador **new** para criar o objeto String. O Java chama **implicitamente** esse operador.

- A classe String possui mais de 60 métodos para manipulação de strings.
- Alguns dos métodos mais usados são definidos a seguir:



Método	Descrição
char charAt(int n)	Retorna o caracter na n-ésima posição (começa a partir do ZERO)
int compareTo(String outra)	Compara a string com outra. Retorna <1, 0 ou >1 caso a string seja menor, igual ou maior que a outra, respectivamente
boolean equals(String outra)	Compara a string com outra (case-sensitive)
boolean equalsIgnoreCase(String outra)	Compara a string com outra ignorando maiúsculas e minúsculas
int indexOf(char ch)	Retorna a primeira ocorrência de ch na string ou -1 se não existir
int indexOf(String str)	Retorna a primeira ocorrência de str na string ou -1 se não existir
int indexOf(String str, int n)	Retorna a primeira ocorrência de str na string começando a busca na posição n
boolean isEmpty()	Verifica se a string é vazia
int lastIndexOf(char ch)	Retorna a última ocorrência de ch na string ou -1 se não existir
int lastIndexOf(String str)	Retorna a última ocorrência de str na string ou -1 se não existir
int lastIndexOf(String str, int n)	Retorna a última ocorrência de str na string começando a busca na posição n
int length()	Retorna o tamanho da string em caracteres
String replace(char velho, char novo)	Substitui todas as ocorrências do caracter velho pelo novo
String substring(int inicio)	Retorna a substring que começa na posição início até o fim da string
String substring(int inicio, int fim)	Retorna a substring da posição início até fim-1
String toLowerCase()	Retorna a string convertida para letras minúsculas
String toUpperCase()	Retorna a string convertida para letras maiúsculas
String trim()	Retira os espaços em branco no início e no fim da string
static String format(String fmt, ...)	Formata as variáveis (igual ao printf) e retorna a string formatada.

- ❑ Strings, em Java, são **imutáveis**.
- ❑ Isso significa dizer que, uma vez criadas, não podemos alterar o conteúdo de uma string.
- ❑ Quando usamos um método que "altera" a string, na realidade estamos **criando um outro objeto** na memória.
- ❑ Exemplo:

```
String disciplina = "Programação Orientada a Objetos";
```

"Programação Orientada a Objetos"

```
String maiuscula = disciplina.toUpperCase();
```

"PROGRAMAÇÃO ORIENTADA A OBJETOS"

```
String outra = maiuscula.replace(' ', '-');
```

"PROGRAMAÇÃO-ORIENTADA-A-OBJETOS"

□ Outro exemplo:

```
String disciplina = "Programação";
```

```
disciplina = disciplina + " Orientada";
```

```
disciplina = disciplina + " a";
```

```
disciplina = disciplina + " Objetos";
```

" Programação"

□ Outro exemplo:

```
String disciplina = "Programação";
```

```
disciplina = disciplina + " Orientada";
```

```
disciplina = disciplina + " a";
```

```
disciplina = disciplina + " Objetos";
```

" Programação"

" Programação Orientada"

□ Outro exemplo:

```
String disciplina = "Programação";
```

```
disciplina = disciplina + " Orientada";
```

```
disciplina = disciplina + " a";
```

```
disciplina = disciplina + " Objetos";
```

" Programação"

" Programação Orientada"

"Programação Orientada a"

□ Outro exemplo:

```
String disciplina = "Programação";
```

```
disciplina = disciplina + " Orientada";
```

```
disciplina = disciplina + " a";
```

```
disciplina = disciplina + " Objetos";
```

" Programação"

" Programação Orientada"

"Programação Orientada a"

"Programação Orientada a Objetos"

A JVM irá descartar automaticamente esses objetos quando necessário.

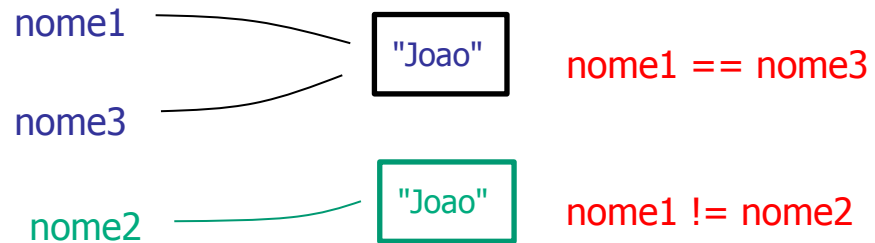
- Ou seja, se alteramos com muita frequência uma string, temos uma perda de desempenho relacionada à cópia dessa string para um novo objeto. E quanto maior a string, pior o desempenho.
- Por isso, para programas que trabalham intensamente com strings use a classe [StringBuilder](#).

- Um detalhe muito importante da manipulação de strings é em relação à comparação de strings.
- Como strings são objetos, se usarmos os operadores relacionais (`==`, `!=`, `>`, `<`, `>=` ou `<=`) estaremos comparando as referências para as strings e não o conteúdo das strings.
- Por isso **não** podemos usar os operadores relacionais (`==`, `!=`, `>`, `<`, `>=` ou `<=`) para comparar strings.
- Para realizar a comparação do conteúdo devemos usar os métodos:
 - `equals()`
 - `equalsIgnoreCase()`
 - `compareTo()`
 - `compareToIgnoreCase()`

Exemplo 1:

```
Scanner t = new Scanner(System.in);  
String nome1 = t.nextLine();  
String nome2 = t.nextLine();  
String nome3 = nome1;
```

```
if (nome1 == nome2)  
    System.out.println("Igual");  
else  
    System.out.println("Diferente");
```

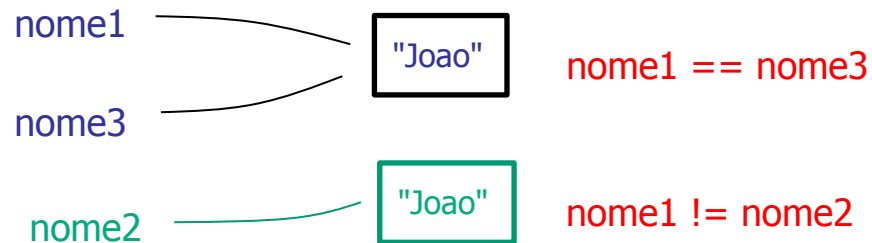


Quando comparamos
nome1 == nome2
estamos comparando se a **área de memória referenciada** é a mesma e
não se o **conteúdo** é o mesmo.
Nesse caso vai imprimir "Diferente"

Exemplo 2:

```
Scanner t = new Scanner(System.in);  
String nome1 = t.nextLine();  
String nome2 = t.nextLine();  
String nome3 = nome1;
```

```
if (nome1.equals(nome2) )  
    System.out.println("Igual");  
else  
    System.out.println("Diferente");
```



Com o método **equals()** estamos comparando os conteúdos. Nesse caso vai imprimir "Igual"

- Exercício 27: crie um programa Java que lê duas strings e implementa os seguintes itens:
 - a) Imprimir a quantidade de vogais da primeira string.
 - b) Criar uma outra string que é idêntica à primeira, mas sem as vogais.
 - c) Criar uma outra string com as letras que fazem parte da primeira string (sem repetições)
 - d) Criar um outra string com os caracteres comuns entre a primeira e a segunda string.
 - e) Criar uma outra string com as vogais que fazem parte da primeira string (sem repetições)
- Nas letras **b** a **e**, a string criada deverá ser impressa.