```
1:  package <nome_package>     CLASSE SEM PACKAGE -> SE RAIZ DO PROJETO
2:
3:  import <outros_packages>
4:
5:  public class NomeClasse {
6:     <atributos>
7:     <construtores>
8:     <outros métodos>
9:  }
```

Classe Principal

String...args

essa variável pode mudar o nome

```
1:  public class ClassePrincipal {
2:     public static void main (String[ ] args) {
3:        //estrutura do main
4:     }
5:  }
```

**Atenção**: Método **static**

Um método que é modificado com a palavra reservada static é invocado sem uma referência a um objeto particular. O nome da classe é usado em seu lugar. Esses métodos são chamados de métodos da classe.

```
C:\workspace\java Hello.java
```

```
javac -cp <caminho da classe> -d <destino da classe compilada> <nome da classe>.java
```

```
java -cp <caminho da classe> <nome do package>.<nome da classe>
```

```
1:  public class Hello {
2:     public static void main(String[] args) {
3:        System.out.println("Hello world");
4:     }
5:  }
```

inteiros

fracionários

| Tipo | Alocação | Exemplo |
|---|---|---|
| byte | 8 bits | 123 |
| short | 16 bits | 1234 |
| **int** | **32 bits** | **12345** |
| long | 64 bits | 123L |
| float | 32 bits | 123.45f |
| **double** | **64 bits** | **123.45, 123.45d** |
| char | 16 bits | '\u004a', 'J', 49 |
| boolean | 1 bit | true, false |

UTF-16

1 - V
0 - F

```
long o = 076554L //octal
long h = 0xaabf4 //hexadecimal
long b = 0b01_11_10_10_10 //binário
long l = 123_456_789.00 //decimal
```

A partir do Java 7, o underline (_) pode ser utilizado para separar números. Ele pode ser colocado em qualquer posição entre os números para facilitar a leitura. É ignorado pelo compilador.

1_000_000_000

| Operador | Tipo | Operação | Exemplo |
|---|---|---|---|
| = | Binário | Atribuição | `long l = 0L` |
| + | Binário | Adição | `7 + 5 = 12` |
| - | Binário | Subtração | `7 - 5 = 2` |
| * | Binário | Multiplicação | `7 * 5 = 35` |
| / | Binário | Divisão | `35 / 7 = 5` |
| % | Binário | Resto da divisão | `7 % 5 = 2` |
| ++ | Unário | Incremento | `int v = 5;`<br>`v++; // 6` |
| -- | Unário | Decremento | `int v = 5;`<br>`v--; // 4` |
| ? : | Ternário | Lógica | `5 > 7 ? 'S' : 'N'` |

```
char r;
if (5 > 7) {
   r = 'S';
} else {
   r = 'N';
}
```

```
char r = 5 > 7 ? 'S' : 'N';
```

| Operadores | Descrição | Exemplo |
|---|---|---|
| < | menor que | `int i = 3, j = 7`<br>`(i < j) => true` |
| <= | menor ou igual | `int i = 3, j = 3`<br>`(i <= j) => true` |
| > | maior que | `int i = 3, j = 7`<br>`(i > j) => false` |
| >= | maior ou igual | `int i = 3, j = 3`<br>`(i >= j) => true` |
| == | igual | `int i = 3, j = 7`<br>`(i == j) => false` |
| != | diferente | `int i = 3, j = 7`<br>`(i != j) => true` |

| Operadores | Descrição | Exemplo |
|---|---|---|
| && | operador e (AND) | `int i = 3, j = 7`<br>`(i < j && j != 3) => true` |
| \|\| | operador ou (OR) | `int i = 3, j = 3`<br>`(i <= j \|\| j != 3) => true` |
| ! | negação (NOT) | `int i = 3, j = 7`<br>`!(i < j) => false` |

if () else :

```
1:  public static void main(String[] args) {
2:     int hora = 19;
3:     if (hora < 12) {
4:        System.out.println("Bom dia!");
5:     } else if (hora < 18) {
6:        System.out.println("Boa tarde!");
7:     } else {
8:        System.out.println("Boa noite!");
9:     }
10: }
```

switch () case:

```
1:  public static void main(String[ ] args) {
2:     char simbolo = '#';
3:     switch (simbolo) {
4:        case '$' : System.out.println("Dólar");
5:           break;
6:        case '#' : System.out.println("Cerquilha");
7:           break;
8:        case '@' : System.out.println("Arroba");
9:           break;
10:       case '%' : System.out.println("Porcentagem");
11:          break;
12:       default: System.out.println("Símbolo não encontrado!");
13:    }
14: }
```

A instrução switch case faz validação por igualdade. Ela pode ser utilizada com os tipos: **byte**, **short**, **int**, **char**, **String**, **Enum** ou uma das **wrapper classes** correspondentes.

No que a palavra reservada **break** é utilizada no final de cada condição **case**. Ela deve ser usada caso seja necessário impedir a execução das condições case subsequentes.

Loops:

```
1:  public class TesteWhileLoop {
2:
3:     public static void main(String[] args) {
4:        var i = 0;
5:        while (i < 100) {
6:           i++;
7:           System.out.printf("i: %d \n", i);
8:        }
9:     }
10: }
```

Executa de 0 a N vezes

```
1:  public class TesteDoLoop {
2:    public static void main(String[] args) {
3:       var i = 0;
4:       do {
5:          i++;
6:          System.out.printf("i: %d \n", i);
7:       } while (i < 100);
8:    }
9:  }
```

Executa de 1 a N vezes

```
10: public class TesteForLoop {
11:
12:   public static void main(String[] args) {
13:
14:      for (int i = 0, j = 100; i < j; i+=2, j-=3) {
15:         System.out.printf("i: %d, j: %d \n", i, j);
16:      }
17:   }
18:
19: }
```

```
for ( ; ; ){
   //loop infinito
}
```

## Array

```
1:  public class TesteArrayForEach {
2:
3:    public static void main(String[] args) {
4:
5:       String meses[] = {"JAN", "FEV", "MAR", "ABR", "MAI", "JUN", "JUN", "AGO",
6:             "SET", "OUT", "NOV", "DEZ"};
7:       double valores[] = new double[5];
8:       valores[0] = 1.5;
9:       valores[1] = 2.5;
10:      valores[2] = 3.5;
11:      valores[3] = 4.5;
12:      valores[4] = 5.5;
13:
14:      for (String m : meses) {
15:         System.out.printf("%s ", m);
16:      }
17:
18:      System.out.println("\n");
19:
20:      for (double v : valores) {
21:         System.out.println(v);
22:      }
23:   }
24: }
```

[0.0] [0.0] [0.0] [0.0] [0.0]

1: declaração de uma variável para receber os elementos do array durante a iteração
2: o array que será percorrido

```
public static void main(String...args) {
    ArrayList<Integer> numeros = new ArrayList({
        add(1); add(2);add(3);add(4);add(5);
    });
    for (Integer i : numeros) {
       //código para kda numero
    }
    numeros.forEach( n -> {
      //código para kda numero
    });
}
```

## String : Classe (array de char)

```
1:  public class TesteString {
2:
3:     public static void main(String[] args) {
4:        String s1 = "Curso";
5:        String s2 = "Java";
6:        String s3 = "Avançado";
7:        String s4 = "Ka";
8:
9:        String frase = s1 + " " + s2 + " " + s3 + " " + s4;
10:
11:       System.out.printf("Frase: %s \n", frase);
12:
13:       String naõFaçaIsso = new String("Má prática");
14:    }
15:
16: }
```

## StringBuilder

```
1:  public class TesteStringBuilder {
2:
3:     public static void main(String[] args) {
4:        String s1 = "Curso";
5:        String s2 = "Java";
6:        String s3 = "Avançado";
7:        String s4 = "Ka";
8:
9:        StringBuilder frase = new StringBuilder(s1).append(" ")
10:              .append(s2).append(" ")
11:              .append(s3).append(" ")
12:              .append(s4);
13:
14:       frase.insert(11, "11 ");
15:       System.out.printf("Frase: %s \n", frase);
16:
17:    }
18:
19: }
```

## Entidade

```
1:  public class Funcionario {
2:     //atributos
3:     private int codigo;
4:     private String nome;
5:     private String setor;
6:     private double salario;
7:     private boolean ativo;
8:
9:     //construtor padrão
10:    public Funcionario() {
11:
12:    }
13:
14:    //getters e setters
15:    public void setCodigo(int codigo) {
16:       this.codigo = codigo;
17:    }
18:
19:    public int getCodigo() {
20:       return this.codigo;
21:    }
22:    //... demais métodos
23: }
```

# Métodos Construtores:

new Construtor()

Sem argumentos => Construtor padrão
fornecido sempre q a classe não tem construtores

```
1:    public Funcionario(int codigo, String nome, String setor, double salario, boolean ativo) {
2:       this.codigo = codigo;
3:       this.nome = nome;
4:       this.setor = setor;
5:       this.salario = salario;
6:       this.ativo = ativo;
7:    }
```

```
1:  public class TesteFuncionario {
2:
3:     public static void main(String[] args) {
4:        Funcionario douglas = new Funcionario(1, "Douglas Oliveira", "TI",
5:              12_000.00, true);
6:        douglas.aumentaSalario(500.00);
7:        System.out.printf("Salário antes do desconto: %.2f \n", douglas.getSalario());
8:
9:        douglas.aplicaDesconto(.05);
10:       System.out.printf("Salário após o desconto: %.2f \n", douglas.getSalario());
11:
12:    }
13:
14:    /*
15:     output:
16:       Salário antes do desconto: 12500,00
17:       Salário após o desconto: 11875,00
18:     */
19: }
```

Outros métodos:

```
8:    public double getSalario() {
9:       return salario;
10:   }
11:
12: public void aumentaSalario(double valor) {
13:       if (valor > 0) {
14:          salario += valor;
15:       }
16:   }
17:
18:   public void aplicaDesconto(double porcentagem) {
19:       salario -= salario * porcentagem;
20:   }
```

Para importar todas as classes de um pacote podemos utilizar um podemos usar o operador curinga "*":

```
import java.util.*;
```

Ou para importar uma classe específica podemos dar o nome da classe:

```
import java.util.Date;
```

**Atenção**: Importação ambígua

Quando usamos o "*" para importar as classes podemos cair em uma situação de ambiguidade, por exemplo:

```
import java.sql.*
import java.util.*
```

java.sql.Statement;

java.util.Date;

Em ambos os pacotes temos a classe **Date**, ou seja, se instanciarmos uma data usando new teremos um erro de compilação pois o Java não conseguirá determinar qual **Date** estamos nos referindo:

```
Date date = new Date(); //erro de compilação
```

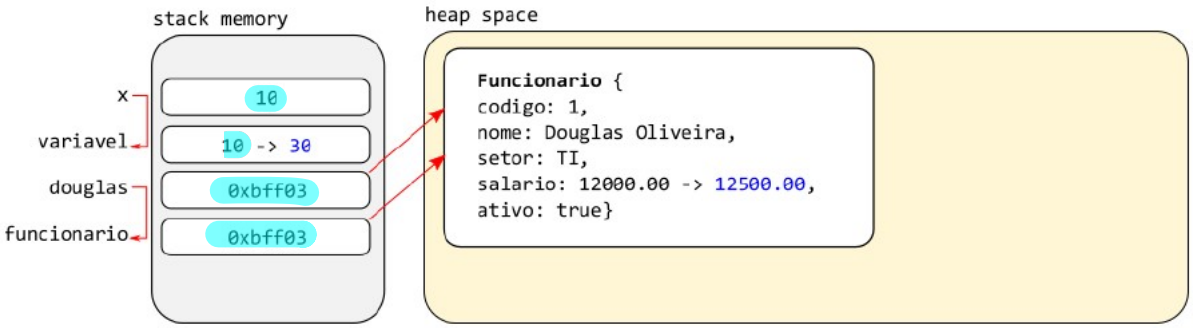
Para resolver esse problema teremos que usar o full qualified name (FQN) da classe:

```
java.util.Date date = new java.util.Date();
```

Para importar todos os membros estáticos e uma classe (static) usamos o ***static import***:

```
import static java.lang.Math.*;
```

pow(3,2);
random();
max(), min()...



# Capítulo 2

| Modificador | Visível para | | | |
|---|---|---|---|---|
| | a própria classe | classes do pacote | classes filhas | todas |
| **private** | Sim | Não | Não | Não |
| **sem modificador** | Sim | Sim | Não | Não |
| **protected** | Sim | Sim | Sim (extends) | Não |
| **public** | Sim | Sim | Sim | Sim |

atributos (variáveis da classe) & métodos

```
1:  public class Produto {
2:
3:     public Integer codigo;
4:     public String descricao;
5:     public String unidade;
6:     public Double valor;
7:
8:  }
```

```
1:  public class TesteProduto01 {
2:
3:     public static void main(String[] args) {
4:        Produto produto = new Produto(); //construtor vazio
5:        produto.codigo = -100; //código negativo
6:        produto.descricao = null; //sem descrição
7:        produto.unidade = "XYZ"; //unidade que não existe
8:        produto.valor  = -1_000.00; //valor negativo
9:     }
10:
11: }
```

Encapsulamento:

```
1:  public class Produto {
2:
3:     private static int ultimoCodigo = 0;
4:
5:     private Integer codigo;
6:     private String descricao;
7:     private String unidade;
8:     private Double valor;
9:
10:    public Produto(String descricao, String unidade, Double valor) {
11:       this.codigo = ++ultimoCodigo;
12:       this.descricao = descricao;
13:       this.unidade = unidade;
14:       this.valor = valor;
15:    }
16: }
```

getters:

```
public <tipo atributo> get<nome atributo> {
      return <nome atributo>;
}
```

```
public String getDescricao() { return descricao; }
```

setters:

```
public void set<nome atributo>(<tipo atributo> <nome atributo>) {
      [validação if/switch case...]
      this.<nome atributo> = <nome atributo>
}
```

```
public void setDescricao(String descricao) {
   if (!descricao.isBlank()) {
      this.descricao = descricao;
   }
}
```

```java
public class Produto {

   private static int ultimoCodigo = 0;
   private Integer codigo;
   private String descricao;
   private String unidade;
   private Double valor;

   public Produto(String descricao, String unidade, Double valor) {
      this.codigo = ++ultimoCodigo;
      setDescricao(descricao);
      setUnidade(unidade);
      setValor(valor);
   }

   public Integer getCodigo() { return codigo; }

   public String getDescricao() { return descricao; }

   public void setDescricao(String descricao) {
      if (!descricao.isBlank()) {
         this.descricao = descricao;
      }
   }

   public String getUnidade() { return unidade; }

   public void setUnidade(String unidade) {
      switch (unidade) {
         case "KG": case "LT": case"CX": case "MT": case "UN":
            this.unidade = unidade;
         default:
            this.unidade = "UN";
      }
   }

   public Double getValor() { return valor; }

   public void setValor(Double valor) {
      if (valor > 0) {
         this.valor = valor;
      }
   }
}
```

## Sobrecarga de construtor:

```java
public class Produto {

   private static int ultimoCodigo = 0;

   private Integer codigo;
   private String descricao;
   private String unidade;
   private Double valor;

   public Produto(String descricao, String unidade, Double valor) {
      this.codigo = ++ultimoCodigo;
      setDescricao(descricao);
      setUnidade(unidade);
      setValor(valor);
   }

   public Produto(String descricao, Double valor) {
      this(descricao, "UN", valor);
   }
}
```

- `Produto(String, String, Double)` ou
- `Produto(String, Double)`

1. A quantidade de argumentos
2. O tipo dos argumentos
3. A ordem dos argumentos de acordo com o tipo