

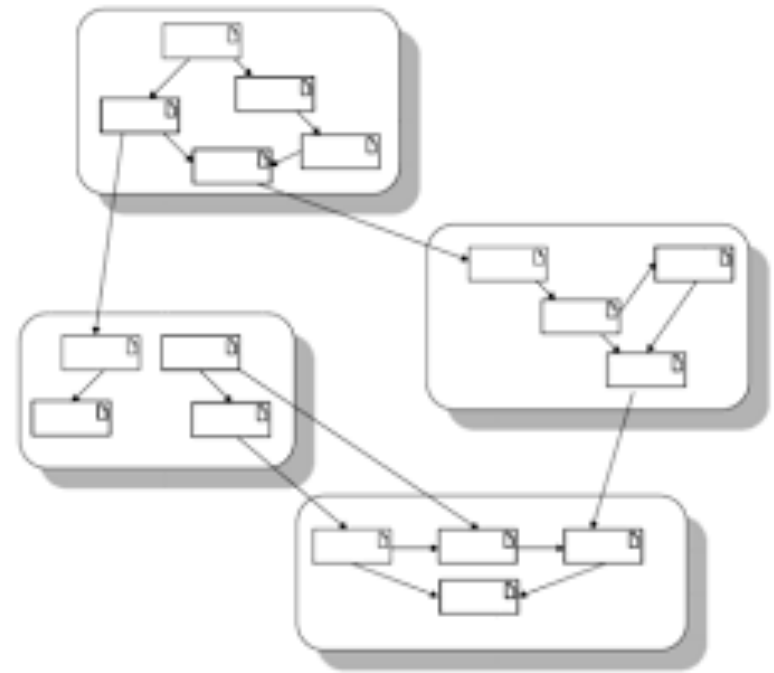
Modularidade

Métodos estáticos

UA.DETI.POO

Programação modular

- ❖ Organização de programas como módulos independentes.
- ❖ Porquê? → Mais fácil de compartilhar e reutilizar o código para criar programas maiores.
- ❖ Em Java podemos considerar como módulo cada ficheiro `.java`.
- ❖ Cada ficheiro `.java` contém uma classe (pública).



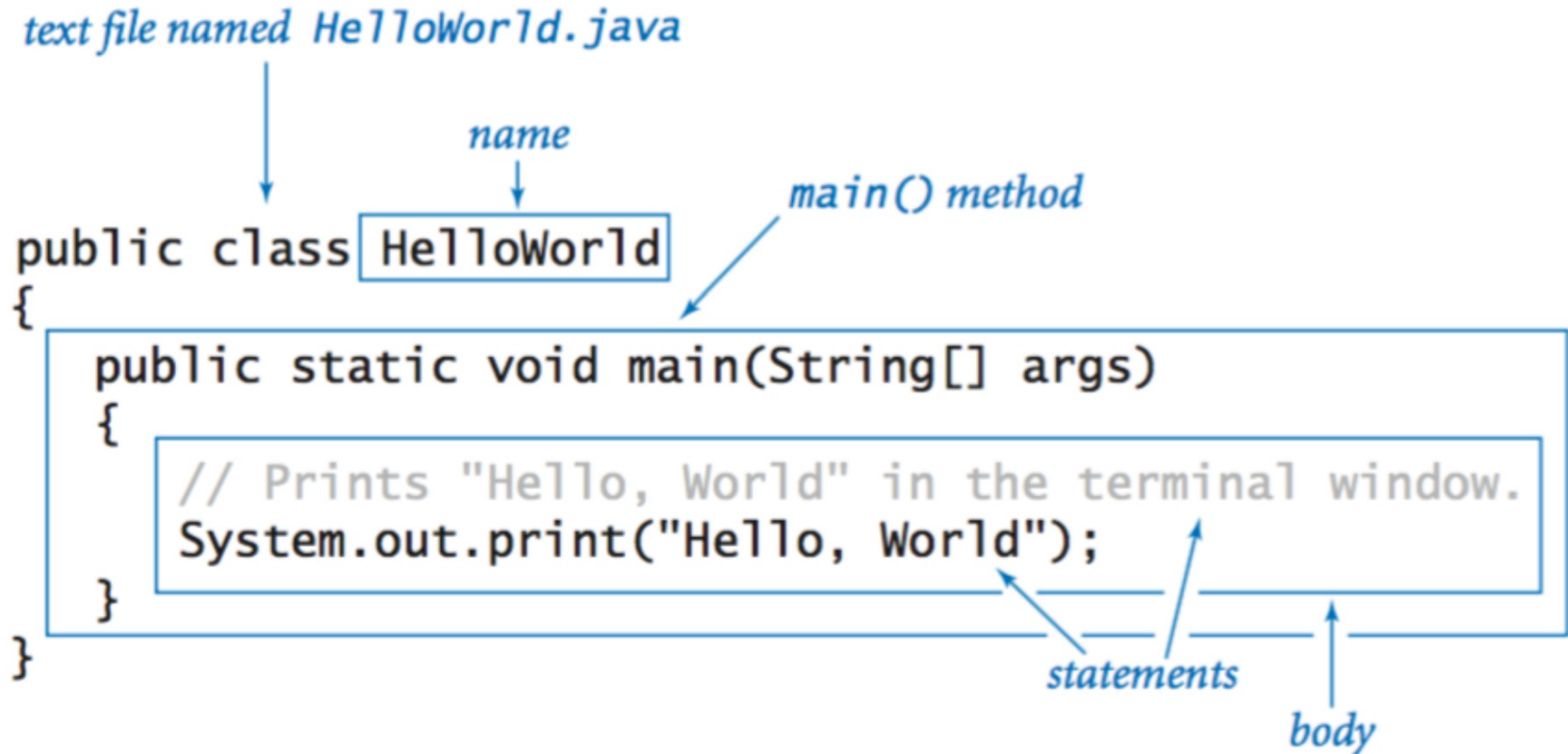
Conceito básico de classe

- ❖ Definição duma classe (ficheiro `Exemplo.java`):

```
public class Exemplo {  
    // dados  
    // métodos  
}
```

- ❖ O ficheiro `Exemplo.java` deve conter uma classe pública denominada `Exemplo`.
 - Devemos usar uma nomenclatura do tipo `Person`, `SomeClass`, `SomeLongNameForClass`, ... *camel case*
 - Java é uma linguagem *case-sensitive* (i.e. `Exemplo` != `exemplo`)
- ❖ Esta classe deve ser declarada como `public`

Classe principal e método main



Funções/métodos estáticos

❖ Uma função

- Realiza uma tarefa.
- Tem zero ou mais argumentos de entrada.
- Retorna zero ou um valor de saída.

❖ Aplicações

- Os cientistas usam funções matemáticas para calcular fórmulas.
- Os programadores usam funções para construir programas modulares.
- Vamos usá-las para ambos os objetivos.

❖ Exemplos

`Math.random()`, `Math.abs()`, `Integer.parseInt()`
`System.out.println()`, `main()`

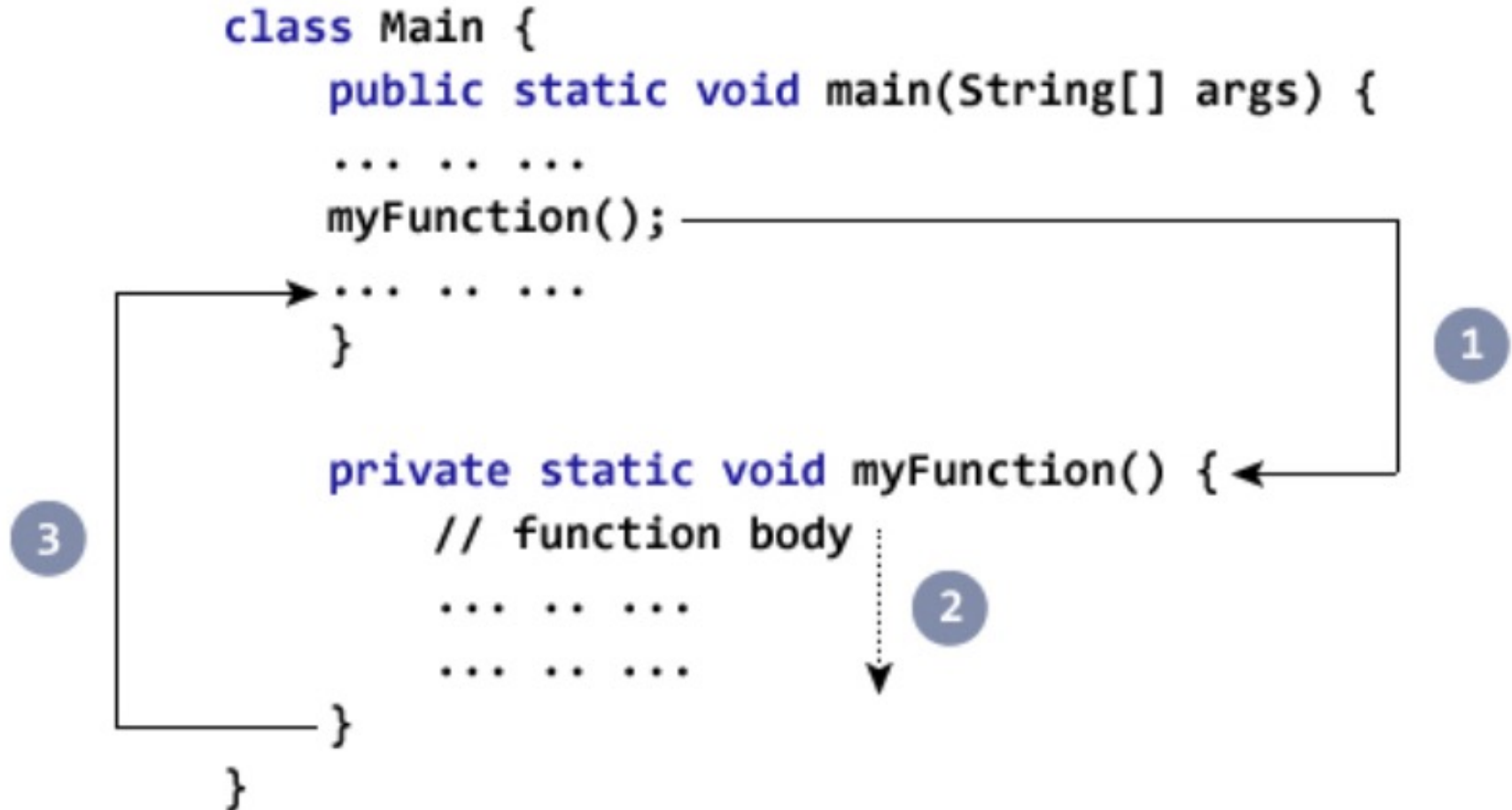
Métodos estáticos

- ❖ Para implementar uma função (método estático), precisamos de
 - Criar um nome
 - Declarar o tipo e o nome do(s) argumento(s)
 - Especificar o tipo para o valor de retorno
 - Implementar o corpo do método
 - Terminar com a declaração de retorno

```
public static void myFunction() {  
    System.out.println("My Function called");  
}
```

```
public static double doisXQuadrado(double x) {  
    return 2*x*x;  
}
```

Execução



<https://www.programiz.com/java-programming/methods>

Exemplos

```
public class Testes {  
  
    public static void main(String[] args) {  
        System.out.println("About to encounter a method.");  
        // method call  
        myMethod();  
        System.out.println("Method was executed successfully!");  
    }  
  
    // method definition  
    private static void myMethod() {  
        System.out.println("Printing from inside myMethod()!");  
    }  
  
}
```


Exemplos

```
public class Testes {  
  
    public static int getIntegerSum(int i, int j) {  
        return i + j;  
    }  
  
    public static int multiplyInteger(int x, int y) {  
        return x * y;  
    }  
  
    public static void main(String[] args) {  
        System.out.println("10 + 20 = " + getIntegerSum(10, 20));  
        System.out.println("20 x 40 = " + multiplyInteger(20, 40));  
    }  
}
```

```
10 + 20 = 30  
20 x 40 = 800
```

Exemplos

```
public class Testes {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 5; i++) {  
            // method call  
            int result = getSquare(i);  
            System.out.println("Square of " + i + " is : " + result);  
        }  
    }  
  
    private static int getSquare(int x) {  
        return x * x;  
    }  
}
```

```
Square of 1 is : 1  
Square of 2 is : 4  
Square of 3 is : 9  
Square of 4 is : 16  
Square of 5 is : 25
```

java.lang.Math

- ❖ A classe *Math* contém métodos estáticos para executar operações numéricas básicas
 - funções exponenciais, logarítmicas, de raiz quadrada e trigonométricas.

Modifier and Type	Method and Description
static double	abs (double a) Returns the absolute value of a double value.
static float	abs (float a) Returns the absolute value of a float value.
static int	abs (int a) Returns the absolute value of an int value.
static long	abs (long a) Returns the absolute value of a long value.
static double	acos (double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through π .

<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

java.lang.Math

❖ Funções gerais

`Math.abs()` → valor absoluto

`Math.ceil(n)` → valor inteiro maior mais próximo de n { Returns double

`Math.floor(n)` → valor inteiro menor mais próximo de n

`Math.min()` { min/máx entre 2 valores `Math.min(23, -1)`

`Math.max()`

`Math.round()` { arredondamento normal

`Math.random()` { número random entre 0.0 e 1.0

❖ Funções exponenciais, logarítmicas

`Math.exp(n)` 10^n

`Math.log()`

`Math.log10(n)` $\log_{10} n$

`Math.pow()` → `Math.pow(2, 3) = 23`

`Math.sqrt()`

java.lang.Math

❖ Funções trigonométricas

```
Math.PI  
Math.sin()  
Math.cos()  
Math.tan()  
Math.asin()  
Math.acos()  
Math.atan()  
Math.atan2()  
Math.sinh()  
Math.cosh()  
Math.tanh()  
Math.toDegrees()  
Math.toRadians()
```

Exemplos

```
public class Testes {  
  
    public static void main(String[] args) {  
        double x = 2.75;  
        System.out.println("número aleatório = " + Math.random());  
        System.out.println("x = " + x);  
        System.out.println("sin = " + Math.sin(x));  
        System.out.println("cos = " + Math.cos(x));  
        System.out.println("sqrt = " + Math.sqrt(x));  
        System.out.println("round = " + Math.round(x));  
        System.out.println("ceil = " + Math.ceil(x));  
    }  
}
```

```
número aleatório = 0.7283141219266507  
x = 2.75  
sin = 0.38166099205233167  
cos = -0.9243023786324636  
sqrt = 1.6583123951777  
round = 3  
ceil = 3.0
```

Strings

UA.DETI.POO

A classe String

- ❖ A classe `java.lang.String` facilita a manipulação de cadeias de caracteres.
- ❖ Exemplo:

```
String s1 = "java"; // creating string by java string literal
char ch[] = { 's', 't', 'r', 'i', 'n', 'g', 's' };
String s2 = new String(ch); // converting char array to string
System.out.println(s1);
System.out.println(s2);
```

java
strings

Concatenação de Strings

❖ Concatenação de Strings

```
String data = " feve" + "reiro ";  
data = 10 + data;  
data += "de " + 2019;  
System.out.println(data);
```

- ❖ Os objetos do tipo String são imutáveis (constantes).
 - Todos os métodos cujo objetivo é modificar uma String, na realidade constroem e devolvem uma String nova
 - A String original mantém-se inalterada.
- Quantos objetos String existem no código acima?

Concatenação de Strings

❖ Utilização alternativa do tipo StringBuilder

```
StringBuilder sb = new StringBuilder();  
sb.append(10);  
sb.append(" feve");  
sb.append("reiro ");  
sb.append("de ");  
sb.append(2019);  
String data = sb.toString();  
System.out.println(data);
```

10 fevereiro de 2019

Métodos da class String

- ❖ Esta classe apresenta um conjunto de métodos que permitem realizar muitas operações sobre texto.

char	charAt(int index) Returns the char value at the specified index.
int	codePointAt(int index) Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index) Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.
int	compareTo(String anotherString) Compares two strings lexicographically.
int	compareToIgnoreCase(String str) Compares two strings lexicographically, ignoring case differences.
String	concat(String str) Concatenates the specified string to the end of this string.
boolean	contains(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.

Comprimento e acesso a caracteres

- ❖ O comprimento (número de caracteres) de uma String pode ser determinado com o método `length`.
- ❖ O acesso a um carater é feito com o método `charAt (int index)`.
- ❖ Exemplo:

```
String s1 = "Universidade de Aveiro";  
System.out.println(s1.length());  
for (int i=0; i < s1.length(); i++ )  
    System.out.print(s1.charAt(i) + ", ");
```

22

U, n, i, v, e, r, s, i, d, a, d, e, , d, e, , A, v, e, i, r, o,

Comparação de Strings

❖ Alguns métodos

- equals, equalsIgnoreCase, compareTo

❖ Exemplos:

```
String s1 = "Aveiro";  
String s2 = "aveiro";
```

```
System.out.println(s1.equals(s2) ? "Iguais" : " Diferentes");  
System.out.println  
    (s1.equalsIgnoreCase(s2) ? "Iguais" : " Diferentes ");  
System.out.println(s1.compareTo(s2));  
    // <0 (s1 menor), 0(iguais), >0 (s1 maior)
```

Comparação de subStrings

- ❖ Podemos analisar partes de uma String
 - contains, substring, startsWith, endsWith, ...

❖ Exemplos:

```
String s1 = "Aveiro";  
String s2 = "aveiro";
```

```
System.out.println(s1.contains("ve")); // true  
System.out.println(s1.substring(1, 3)); // ve  
System.out.println(s1.startsWith("ave")); // false  
System.out.println(s1.endsWith("ro")); // true
```

Formatação de Strings

- ❖ O método `format` retorna uma String nova formatada de acordo com especificadores de formato.

```
long segundos = 347876;  
String s1 =  
String.format("%02d horas, %02d minutos e %02d segundos\n",  
    segundos / 3600,  
    (segundos % 3600) / 60,  
    segundos % 60);  
System.out.println(s1);
```

```
00      00      00  
96 horas, 37 minutos e 56 segundos
```

<https://docs.oracle.com/javase/tutorial/java/data/numberformat.html>

Formatação de Strings

❖ `System.out.printf` é um método, alternativo ao `System.out.print`, que utiliza formatação.

❖ Exemplo:

```
long segundos = 347876;  
System.out.printf("%02d horas, %02d minutos e %02d segundos\n",  
    segundos / 3600,  
    (segundos % 3600) / 60,  
    segundos % 60);
```

96 horas, 37 minutos e 56 segundos

<https://docs.oracle.com/javase/tutorial/java/data/numberformat.html>

Expressões regulares (regex)

- ❖ Permitem definir padrões que podem ser procurados em Strings.
 - A lista completa de construções suportadas está descrita na documentação da classe `java.util.regex.Pattern`.
- ❖ O método `matches` da classe `String` verifica se uma `String` inclui um dado padrão.
- ❖ Exemplos:

```
String s1 = "123";  
System.out.println(s1.matches("\\d{2,4}"));  
// 2-4 dígitos seguidos  
s1 = "abcdefg";  
System.out.println(s1.matches("\\w{3,}"));  
// pelo menos 3 caracteres alfanuméricos
```

2 a 4 dígitos seguidos

true
true

Método split

- ❖ O método `split` separa uma `String` em partes com base numa expressão regular e devolve o vetor de `Strings` resultantes.

```
String frase = "Regular expressions are powerful and "  
              + "flexible text-processing tools.";  
String[] splitResult = frase.split("\\W");  
    // separa com base em caracteres não alfanuméricos  
System.out.println(splitResult.length + " palavras: " +  
                    Arrays.toString(splitResult));  
splitResult = frase.split("ex");  
System.out.println(splitResult.length + " palavras: " +  
                    Arrays.toString(splitResult));
```

9 palavras: [Regular, expressions, are, powerful, and, flexible, text, processing, tools]
4 palavras: [Regular , pressions are powerful and fl, ible t, t-processing tools.]

Alguns exemplos de padrões regex

- `.` qualquer caracter
- `\d` dígito de 0 a 9
- `\D` não dígito `[^0-9]`
- `\s` “espaço”: `[\t\n\x0B\f\r]`
- `\S` não “espaço”: `[^\s]`
- `\w` carater alfanumérico: `[a-zA-Z_0-9]`
- `\W` carater não alfanumérico: `[^\w]`
- `[abc]` qualquer dos caracteres a, b ou c
- `[^abc]` qualquer carater exceto a, b e c
- `[a-z]` qualquer carater entre a-z, inclusive
- `X?` um ou nenhum X
- `X*` nenhum ou vários X
- `X+` um ou vários X

Sumário

- ❖ Modularidade
- ❖ Funções estáticas
- ❖ Classe Math
- ❖ Classe String
- ❖ Regex