Instruções de controlo de fluxo

UA.DETI.POO

Controlo de fluxo num programa

- A ordem de execução das instruções de um programa é normalmente linear
  - uma declaração após a outra, em sequência
- Algumas instruções permitem alterar esta ordem, decidindo:
  - se deve ou não executar uma declaração particular
  - executar uma declaração repetidamente, repetidamente
- Essas decisões são baseadas em expressões booleanas (ou condições)
  - que são avaliadas como verdadeiras ou falsas

Expressões booleanas

- Expressões booleanas retornam `true` ou `false`.
- As expressões booleanas usam operadores relacionais, de igualdade, e lógicos (AND, OR, NOT)

```
==      equal to        // Atenção!! x == y é diferente de x = y
!=      not equal to
<       less than
>       greater than
<=      less than or equal to
>=      greater than or equal to
!       NOT
&&      AND
||      OR
```

- Exemplos

```
x >= 10
(y < z) && (z > t)
```

Tabelas de verdade

- A álgebra booleana é baseada em tabelas de verdade.
- Considerando A e B, por ex: `((y < z) && (z > t))`
  - Ambos têm que ser verdadeiros para a expressão **A && B** ser verdadeira.
  - Basta um ser verdadeiro para a expressão **A || B** ser verdadeira.

a	*!a*
true	false
false	true

a	*b*	*a && b*	*a || b*
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Operador ternário

- O operador ternário (**?:**) é também conhecido como operador condicional.

```
result = testCondition ? valueIfTrue : valueIfFalse
```

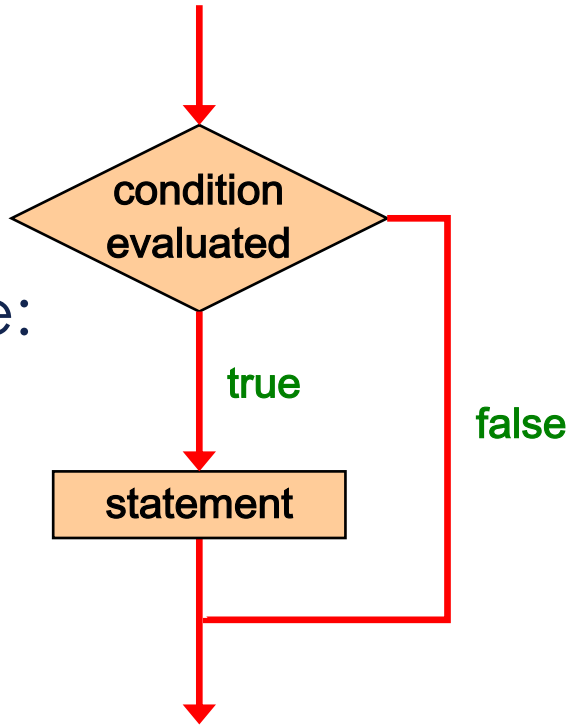
  - Avalia uma expressão (1º operando) e, caso seja true, o resultado é igual ao 2º operando, caso contrário o resultado é igual ao 3º operando.

```
char code = 'F';
boolean capitalLetter = (code >= 'A') && (code <= 'Z');
System.out.println(capitalLetter ? "sim" : "não");

minVal = (a < b) ? a : b;
```

Instruções condicionais

- Em Java existem dois tipos de instruções de decisão/seleção:
  - **if**
  - **switch**



- A instrução if tem o formato seguinte:

```
if (expressãoBooleana)
  // fazer_isto;
else  // opcional
  // fazer_aquilo;
```

Exemplo

```
Scanner sc = new Scanner(System.in);
int number = sc.nextInt();

if (number % 2 == 0)
    System.out.println("O número é par");
else
    System.out.println("O número é ímpar");

sc.close();
```

Instrução de decisão if

- Podemos encadear várias instruções if:

```
if (condição1)
    bloco1;
else if (condição2)
    bloco2;
else
    bloco3;
```

Se um bloco incluir mais que uma instrução, o bloco deve ser delimitado por { .. }.

- Exemplo

```
if (faltas <= 3)
    System.out.println("Pode ir ao exame teórico.");
else if (!regime.equals("T"))
    System.out.println("Reprovado por faltas.");
else {
    System.out.println("Aluno trabalhador sem a/c.");
    System.out.println("Deve fazer exame prático.");
}
```

Instrução de seleção switch

- A instrução switch executa um de entre vários caminhos (case), consoante o resultado de uma expressão

```
switch (expressão) {
    case valor1:
        bloco1;
        break;
    case valor2:
        bloco2;
        break;
    //...
    default:
        blocoFinal;
}
```

O resultado da expressão é pesquisado na lista de alternativas existentes em cada case, pela ordem com que são especificados.

Se a pesquisa for bem sucedida, o bloco de código correspondente é executado. Se houver a instrução break, a execução do switch termina. Caso contrário serão executadas todas as opções seguintes até que apareça break ou seja atingido fim do switch.

Se a pesquisa não for bem sucedida e se o default existir, o bloco de código correspondente (blocoFinal) é executado.

Exemplo

```
switch (category) {
      case 10:
         System.out.println ("a perfect score. Well done.");
         break;
      case 9:
         System.out.println ("well above average. Great.");
         break;
      case 8:
         System.out.println ("above average. Nice job.");
         break;
      case 7:
         System.out.println ("average.");
         break;
      case 6:
         System.out.println ("below average.");
         System.out.println ("See the instructor.");
         break;
      default:
         System.out.println ("not passing.");
   }
```

Exemplo

```
Scanner sc = new Scanner(System.in);
int mes = sc.nextInt();
int dias;
switch (mes)
{  case 4:
   case 6:
   case 9:
   case 11: dias = 30; break;
   case 2:  dias = 28; break;
   default: dias = 31;
}
System.out.println("Mês tem " + dias + " dias");
sc.close();
```

Ciclos

- Por vezes existe a necessidade de executar instruções repetidamente.
  - A um conjunto de instruções que são executadas repetidamente designamos por ciclo.

- Um ciclo pode ser do tipo condicional (**while** e **do…while**) ou do tipo contador (**for**).
  - Normalmente utilizamos ciclos condicionais quando o número de iterações é desconhecido e ciclos do tipo contador quando sabemos à partida o número de iterações.

Ciclo while

- O ciclo **while** executa enquanto a condição do ciclo esteja verdadeira.
  - A condição é avaliada antes de cada iteração do ciclo.

```
while (condição)
    bloco_a_executar;
```

  - Exemplo:

```
Scanner sc = new Scanner(System.in);
int nota = -1;
while ( (nota > 20) || (nota < 0) ) {
    System.out.println("Insira a nota do aluno.");
    nota = sc.nextInt();
}
sc.close();
```

Ciclo do while

- O ciclo **do...while** executa uma primeira vez e só depois verifica se é necessário repetir.
  - A condição é avaliada no fim de cada iteração do ciclo.

```
do
    bloco_a_executar;
while (condição);
```

  - Exemplo:

```
Scanner sc = new Scanner(System.in);
int nota;
do {
    System.out.println("Insira a nota do aluno.");
    nota = sc.nextInt();
} while ( (nota > 20) || (nota < 0) );
sc.close();
```

Ciclo for

- O ciclo **for** é mais geral pois suporta todas as situações de execução repetida.

```
for (inicialização; condição; atualização)
    bloco_a_executar;
```

1. Antes da 1ª iteração, faz a **inicialização** (só uma vez)
2. Depois realiza o teste da **condição**.
   Se for *true* executa o bloco, se for *false* termina
3. No fim de cada iteração, executa a parte de **atualização** e retoma no ponto 2 anterior.

Exemplos

Exemplo 1

```
for (int i = 1 ; i <= 10 ; i++)
    System.out.println(i+" * "+ i +" = " + i*i);
```

Exemplo 2

```
int[] tb = new int[10];
for (int i = 0 ; i < tb.length ; i++)
    tb[i] = i * 2 ;
for (int i = 0 ; i < tb.length ; i++)
    System.out.print(tb[i] + ", ");
```

```
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
6 * 6 = 36
7 * 7 = 49
8 * 8 = 64
9 * 9 = 81
10 * 10 = 100
```

```
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
```

Ciclo for (sintaxe foreach)

- O ciclo for, quando usado com vetores, pode ter uma forma mais sucinta (foreach).

```
Scanner sc = new Scanner(System.in);
double[] a = new double[5];
for (int i = 0; i < a.length; i++)
    a[i] = sc.nextDouble();

for (double el : a)
    System.out.println(el);

sc.close();
```

Instruções break e continue

- Podemos terminar a execução dum bloco de instruções com duas instruções especiais:
  - **break** e **continue**.

- A instrução **break** permite a saída imediata do bloco de código que está a ser executado.
  - É usada normalmente em switch mas também pode ser usada em ciclos.
- A instrução **continue** permite terminar a execução da iteração corrente, forçando a passagem para a iteração seguinte (i.e. não termina o ciclo).

Instruções break e continue

- Exemplo:

```
public class Testes {
    public static void main(String[] args) {
        int[] numbers = { 10, 20, 30, 40, 50 };
        for (int x : numbers) {
            if (x == 30) {
                break;
            }
            System.out.println(x);
        }
    }
}
```

```
10
20
```

Sumário

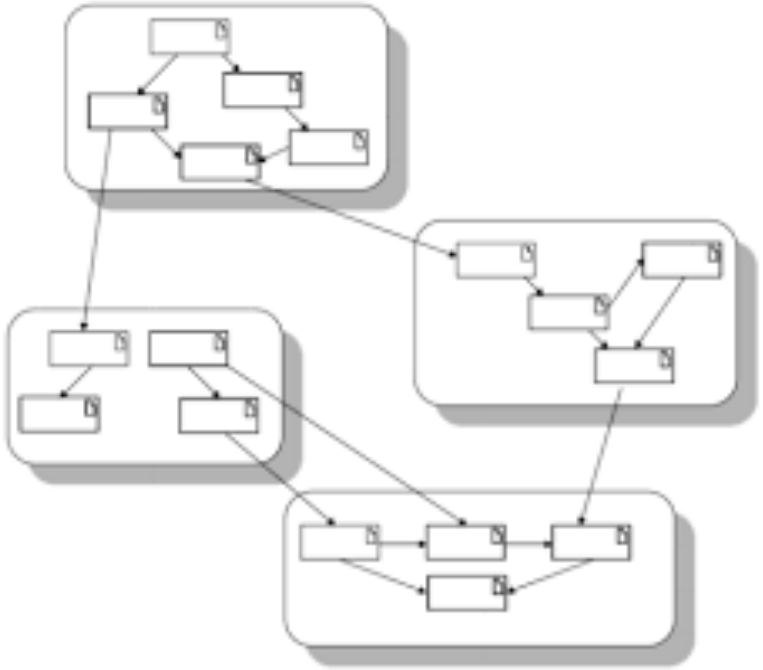
- Instruções condicionais
  - if
  - if .. else
  - switch
- Instruções de ciclos
  - while
  - do … while
  - for

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).



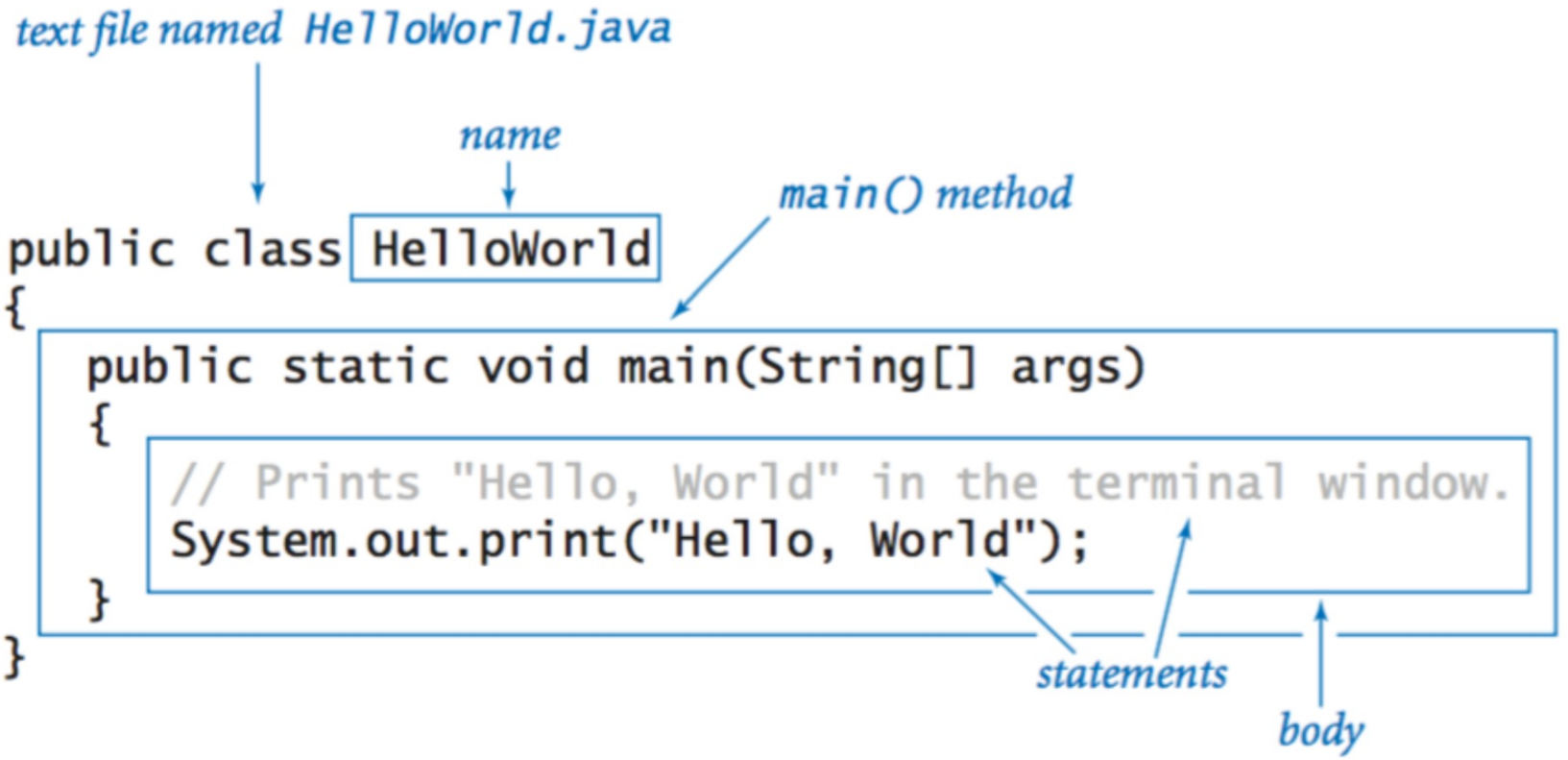
Conceito básico de classe

- 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, …
  - 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()
```

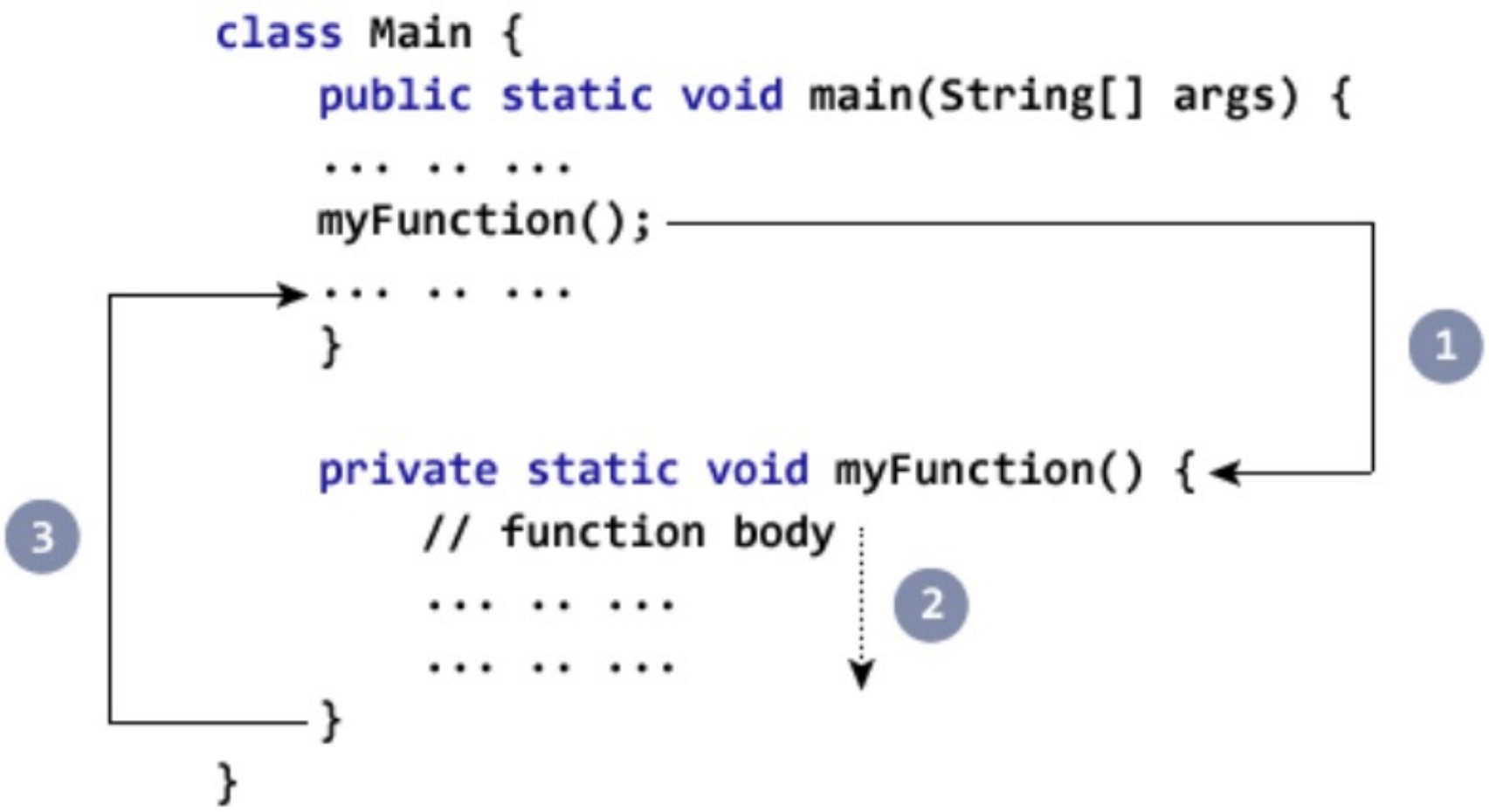
Métodos estáticos

- 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 instrução 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 *pi*.

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

java.lang.Math

- Funções gerais
  - `Math.abs()`
  - `Math.ceil()`
  - `Math.floor()`
  - `Math.min()`
  - `Math.max()`
  - `Math.round()`
  - `Math.random()`
- Funções exponenciais, logarítmicas
  - `Math.exp()`
  - `Math.log()`
  - `Math.log10()`
  - `Math.pow()`
  - `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 carateres.
- 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 to 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 carateres

- O comprimento (número de carateres) 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);
```

```
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 carateres alfanuméricos
```

```
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 carateres 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 carateres 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