

Parte 1: Estruturas de Dados

Parte 2: Persistência de Dados

UA.DETI.POO

Parte 1: Estruturas de Dados

Java Collections

UA.DETI.POO

Prefácio: Tudo são objetos

(ou quase tudo)

Existem dados primitivos

❖ Não são objetos

```
int myNum = 5;           // Integer (whole number)
float myFloatNum = 5.99f; // Floating point number
char myLetter = 'D';     // Character
boolean myBool = true;   // Boolean
String myText = "Hello"; // String
```

https://www.w3schools.com/java/java_data_types.asp

Existem dados primitivos

Data Type	Description
byte	Stores whole numbers from -128 to 127
short	Stores whole numbers from -32,768 to 32,767
int	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	Stores fractional numbers. Sufficient for storing 15 to 16 decimal digits
boolean	Stores true or false values
char	Stores a single character/letter or ASCII values

dados primitivos e classes adaptadoras (“wrappers”)

- ❖ As classes adaptadoras (“wrappers”) são gémeos dos dados primitivos
- ❖ Contêm métodos especializados

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

https://www.w3schools.com/java/java_wrapper_classes.asp

dados primitivos e classes adaptadoras (“wrappers”)

❖ Mas têm gémeos Objects

– Contêm grande `public class Main {`

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

`Integer myInt = 5;`

`Double myDouble = 5.99;`

`Character myChar = 'A';`

`System.out.println(myInt);`

`System.out.println(myDouble);`

`System.out.println(myChar);`

`}`

`}`

Prim

byte

short

int

long

float

doub

boole

char

Character

dados primitivos e classes adaptadoras (“wrappers”)

Module java.base

→ **Package** java.lang

Provides classes that are fundamental to the design of the Java programming language. The most important classes are `Object`, which is the root of the class hierarchy, and `Class`, instances of which represent classes at run time.

Frequently it is necessary to represent a value of primitive type as if it were an object. The wrapper classes `Boolean`, `Character`, `Integer`, `Long`, `Float`, and `Double` serve this purpose. An object of type `Double`, for example, contains a field whose type is `double`, representing that value in such a way that a reference to it can be stored in a variable of reference type. These classes also provide a number of methods for converting among primitive values, as well as supporting such standard

<https://www.tutorialspoint.com/java/lang/>

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/package-summary.html>

dados primitivos e classes adaptadoras (“wrappers”)

Module java.base

Package java.lang

Provides classes that are fundamental to the design of the Java language. The most important classes are Object, which is the root of the class hierarchy, and the wrapper classes, which represent classes at run time.

Frequently it is necessary to represent a value of primitive type in a class. For example, the classes Boolean, Character, Integer, Long, Float, and Double, for example, contains a field whose type is double, and a method that returns a reference to it. That a reference to it can be stored in a variable of reference type. The wrapper classes have a number of methods for converting among primitive values, and for converting them to and from strings.

<https://www.tutorialspoint.com/java/lang/>
<https://docs.oracle.com/en/java/javase/11/api/java.lang/>
[package-summary.html](https://docs.oracle.com/en/java/javase/11/api/java.lang/package-summary.html)

Java.lang Package classes

Java.lang - Home

Java.lang - Boolean

Java.lang - Byte

Java.lang - Character

Java.lang - Character.Subset

Java.lang - Character.UnicodeBlock

Java.lang - Class

Java.lang - ClassLoader

Java.lang - Compiler

Java.lang - Double

Java.lang - Enum

Java.lang - Float

[ack](#)

Exemplo: Int e Integer

Class Declaration

Following is the declaration for **java.lang.Integer** class –

```
public final class Integer
    extends Number
        implements Comparable<Integer>
```

Learn **Java** in-depth with real-world projects through our **Java certification course**. Enroll and become a certified expert to boost your career.

Field

Following are the fields for **java.lang.Integer** class –

- **static int MAX_VALUE** – This is a constant holding the maximum value an int can have, $2^{31}-1$.

https://www.tutorialspoint.com/java/lang/java_lang_integer.htm

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Integer.html>

Exemplo: Int e Integer

Class constructors

Sr.No.	Constructor & Description
1	Integer(int value) This constructs a newly allocated Integer object that represents the specified int value.
2	Integer(String s) This constructs a newly allocated Integer object that represents the int value indicated by the String parameter.

Class methods

Sr.No.	Method & Description
1	static int bitCount(int i) This method returns the number of one-bits in the two's complement binary representation of the specified int value.
2	byte byteValue() This method returns the value of this Integer as a byte.
3	int compareTo(Integer anotherInteger) This method compares two Integer objects numerically.

Collections e Listas

Collections Framework (JCF)

❖ Conjunto de classes, interfaces e algoritmos que representam vários tipos de estruturas de armazenamento de dados

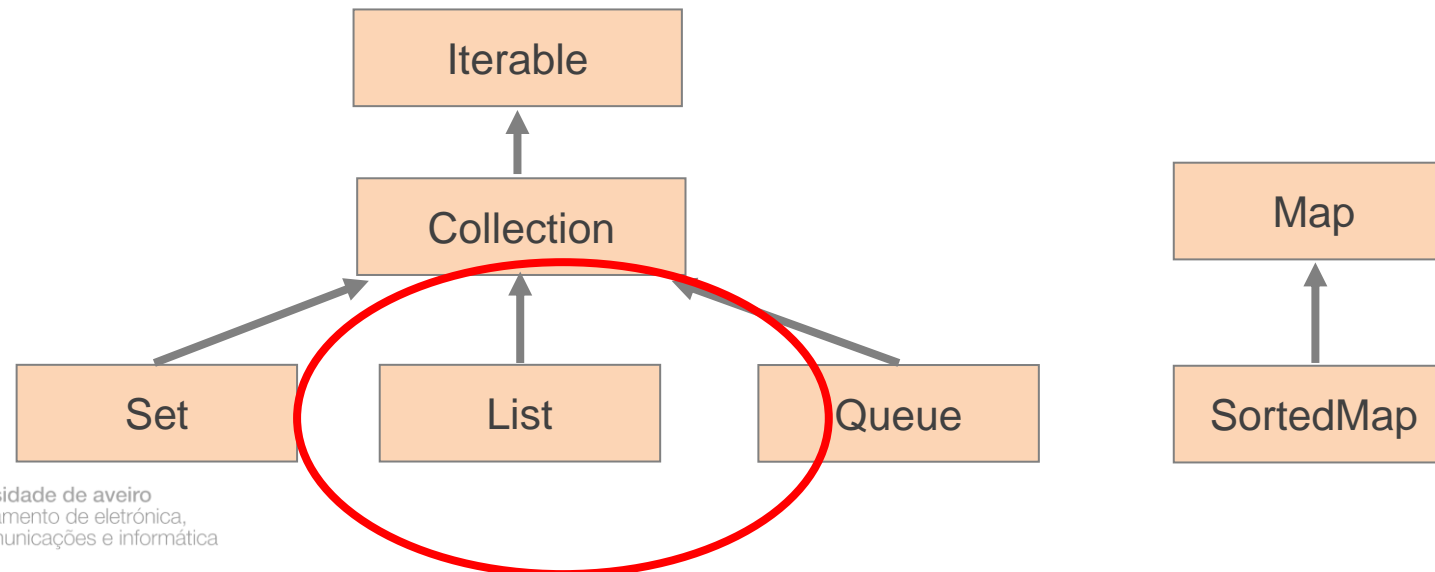
- Listas, Vectors, Pilhas, Árvores, Mapas,...
- Permitem agregar objetos de um tipo paramétrico - os tipos de dados também são um Parâmetro
- Exemplo:

```
ArrayList<String> cidades = new ArrayList<>();  
cidades.add("Aveiro");  
cidades.add("Paris");
```

- Não suportam tipos primitivos (int, float, double,...). Neste caso, precisamos de usar classes adaptadoras (Integer, Float, Double, ...)

Collections

- Conjuntos (Set): sem noção de posição (sem ordem), sem repetição
- Listas (List): sequências com noção de ordem, com repetição
- Filas (Queue): são as filas do tipo *First in First Out*
- Mapas (Map): estruturas associativas onde os objectos são representados por um par chave-valor. 🗨



Listas – Classes

Mais comuns:

- ❖ ArrayList – Array dinâmico
- ❖ LinkedList – Lista ligadas 

Outras:

- ❖ Vector – Array dinâmico
 - (!) *Vector is synchronized. If a thread-safe implementation is not needed, it is recommended to use ArrayList in place of Vector.*
- ❖ Stack
 - extends Vector

Diferenças?

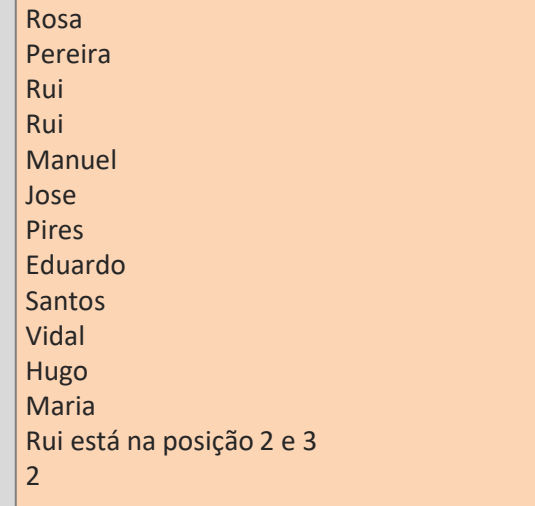
Listas

- ❖ Podem conter duplicados.
- ❖ Para além das operações herdadas de *Collection*, a interface *List* inclui ainda:
 - **Acesso Posicional** — manipulação de elementos baseada na sua posição (índice) na lista
 - **Pesquisa** — de determinado elemento na lista. Retorna a sua posição.
 - **ListIterator** — estende a semântica do *Iterator* tirando partido da natureza sequencial da lista.
 - **Range-View** — execução de operações sobre uma gama de elementos da lista.

`list.subList(fromIndex, toIndex).clear();`

Listas – Exemplo

```
public static void main(String args[]) {  
    String[] str1 = {"Rui", "Manuel", "Jose", "Pires", "Eduardo", "Santos"};  
    String[] str2 = {"Rosa", "Pereira", "Rui", "Vidal", "Hugo", "Maria"};  
    List<String> larray = new ArrayList<>();  
    List<String> llist = new LinkedList<>();  
  
    for (String i: str1 ) larray.add(i);  
    for (String i: str2 ) llist.add(i);  
  
    llist.addAll(llist.size()/2, larray);  
    for (String ele: llist)  
        System.out.println( ele );  
  
    System.out.println("Rui está na posição " +  
        llist.indexOf("Rui") + " e " + llist.lastIndexOf("Rui"));  
  
    llist.set(llist.lastIndexOf("Rui"), "Rui2");  
    System.out.println(llist.lastIndexOf("Rui"));  
}
```



Rosa
Pereira
Rui
Rui
Manuel
Jose
Pires
Eduardo
Santos
Vidal
Hugo
Maria
Rui está na posição 2 e 3
2

Iterar sobre coleções

❖ Iterator



```
List<String> names = Arrays.asList("Ana", "Ze", "Rui");  
Iterator<String> it = names.iterator();  
while (it.hasNext())  
    System.out.println(it.next());
```

❖ ciclo "for each"

```
List<String> names = Arrays.asList("Ana", "Ze", "Rui");  
for (String name : names)  
    System.out.println(name);
```

❖ Método forEach

```
List<String> names = Arrays.asList("Ana", "Ze", "Rui");  
names.forEach(s -> System.out.println(s)); // forEach com lambda  
names.forEach(System.out::println); // forEach com referência de método
```

❖ Stream operations

– *Aggregate operations*

Ler inteiros e coloca-los numa lista

```
# -- !! Python code !! --  
l=[]  
while True:  
    s=input( "v?" )  
    if s=="":  
        break  
    l.append( int( v ) )  
print( "read ", len( l ), " value "  
print(l )
```

<https://chatgpt.com/share/67a270d2-5978-8006-984e-7bd7e1a5ffb5>

Ler inteiros e coloca-los numa lista

```
# -- !! Python code !! --
```

```
l=[]
```

```
→ while True:
```

```
    s=input( "v?")
```

```
→ if s=="":
```

```
    break
```

```
→ l.append( int( v ))
```

```
print( "read ", len( l )," value ")
```

```
print(l )
```

<https://chatgpt.com/share/67a270d27bd7e1a5ffb5>

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

public class Main {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);

        → while (true) {
            System.out.print("v? ");
            String s = scanner.nextLine();
            → if (s.isEmpty()) {
                break;
            }
            → try {
                int v = Integer.parseInt(s);
                list.add(v);
            } catch (NumberFormatException e) {
                System.out.println("Please enter a valid number");
            }
        }

        System.out.println("Read " + list.size() + " values");
        System.out.println(list);

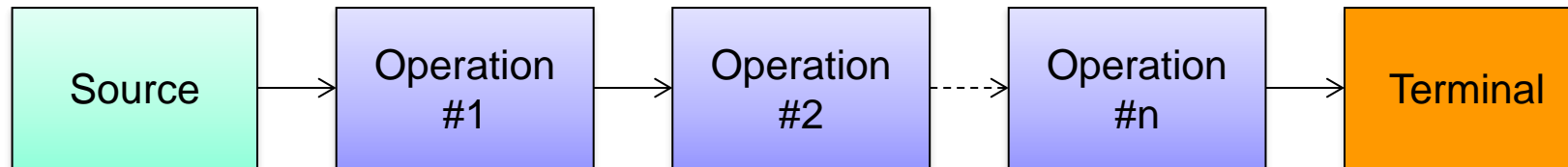
        scanner.close();
    }
}
```

Streams (Básico)

Outra forma de percorrer os listas

Stream Pipeline

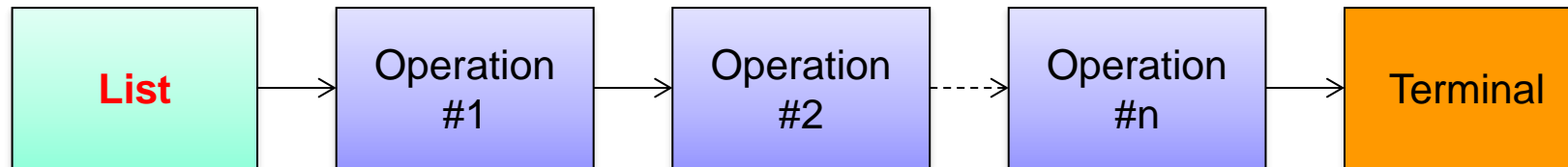
- ❖ (1) Obtain a stream from a source
- ❖ (2) Perform one or more intermediate operations
- ❖ (3) Perform one terminal operation



- ❖ Usage: `Source.Op1.Op2 .. .Terminal`

Transforma lista

- ❖ (1) Obtain a stream from a **List**
- ❖ (2) Perform one or more intermediate operations
- ❖ (3) Perform one terminal operation



- ❖ Usage: `Source.Op1.Op2 .. .Terminal`

Que posso usar

❖ Operações

❖ Com auxilio

- Lambdas



`n -> n>3`

`n -> n+1`

`str -> System.out.println(str)`

`(s1, s2) -> {return s1.compareToIgnoreCase(s2); }`

- Funções (novas ou existentes)

`System.out::println`

`String::compareToIgnoreCase`

Lambda: Sintaxe

- ❖ Uma expressão lambda descreve uma função anónima. Representa-se na forma:
 - (argument) -> (body)
`(int a, int b) -> { return a + b; }`
- ❖ Pode ter zero, um, ou mais argumentos
 - () -> { body }
`() -> System.out.println("Hello World");`
 - (arg1, arg2...) -> { body }
- ❖ O tipo dos argumentos pode ser explicitamente declarado ou inferido
 - (type1 arg1, type2 arg2...) -> { body }
`(int a, int b) -> { return a + b; }`
`a -> return a*a // um argumento – podemos omitir os parêntesis`
- ❖ O corpo (body) pode ter uma ou mais instruções

Lambda: Como usar?

- ❖ Uma expressão lambda não pode ser declarada isoladamente

`(n) -> (n % 2) == 0 // Erro de compilação`

- ❖ Precisamos de outro mecanismo adicional
 - Interfaces funcionais
 - onde as expressões lambda passam a ser implementações de métodos abstratos.
 - O compilador Java converte uma expressão lambda num método da classe (isto é um processo interno).

Lambda: Exemplos

lambda expression	equivalent method
<code>() -> { System.gc(); }</code>	<code>void nn() { System.gc(); }</code>
<code>(int x) -> { return x+1; }</code>	<code>int nn(int x) return x+1; }</code>
<code>(int x, int y) -> { return x+y; }</code>	<code>int nn(int x, int y) { return x+y; }</code>
<code>(String... args) ->{return args.length;}</code>	<code>int nn(String... args) { return args.length; }</code>
<code>(String[] args) -> { if (args != null) return args.length; else return 0; }</code>	<code>int nn(String[] args) { if (args != null) return args.length; else return 0; }</code>

Referências a métodos: 4 variedades

Kind	Syntax/Examples	Equivalent to
Reference to a static method	Class::staticMethod Math::abs Double::compare Math::random	(args) -> Class.staticMethod(args) (x) -> Math.abs(x) (x, y) -> Double.compare(x, y) () -> Math.random()
Reference to an instance method of a particular object	obj::method System.out::println "abcdef"::substring	(args) -> obj.method(args) (s) -> System.out.println(s) (a, b) -> "abcdef".substring(a, b)
Reference to an instance method of arbitrary object of a particular type	Type::method String::compareTo String::strip	(arg1, args) -> arg1.method(args) (s, t) -> s.compareTo(t) (s) -> s.strip()
Reference to a constructor	Class::new File::new int[]::new	(args) -> new Class(args) (name) -> new File(name) (size) -> new int[size]

[Method references \(Java tutorial\)](#)

Alguns exemplos

```
l.stream().filter(n -> n.length()>3)
    .forEach(System.out::println);
```

```
# -- !! Python code !! --
for n in l : if len(n)>3 : print(n)
```

```
List<String> names = people.stream()
    .map(Person::getName)
    .collect(Collectors.toList());
```

```
# -- !! Python code !! --
r=[]
for p in people :
    r.append( p['name'])

r=[ p['name'] for p in people]
```

java.util.stream – Sources

❖ Streams sources include:

- From a `Collection` via the `stream()` and `parallelStream()` methods;
- From an `Array` via `Arrays.stream(Object[])`;
- *and many more (files, random, ..)*

java.util.stream – Terminating operations

❖ Reducers

- reduce(), count(), findAny(), findFirst()

❖ Collectors

- collect()

❖ forEach

❖ iterators

```
// Accumulate names into a List
List<Person> people = ...;
List<String> names = people.stream()
    .map(Person::getName)
    .collect(Collectors.toList());
```

```
# -- !! Python code !! --
names=[ p['name'] for p in person ]
```

Stream.filter

- ❖ Filtering a stream of data is the first natural operation that we would need.
- ❖ Stream interface exposes a filter method that takes in a Predicate that allows us to use lambda expression to define the filtering criteria:

```
List<String> l = Arrays.asList("Ana Maria", "Mariana", "Rui");
```

```
l.stream().filter(n -> n.length()>3)  
    .forEach(System.out::println);
```

```
# -- !! Python code !! --  
for s in l :  
    if len( s) > 3 :  
        print(s )
```


Stream.map



- ❖ The map operations allows us to apply a function that takes in a parameter of one type and returns something else.

```
Stream<Student> map = persons.stream()
    .filter(p -> p.getAge() > 18)
    .map(person -> new Student(person));
```

// other example with Map && Consumer

```
List<String> l = Arrays.asList("Ana", "Ze", "Rui");
l.stream().map(n -> "Nome = " + n)
    .forEach(System.out::println);
```

```
# -- !! Python code !! --
map= [s for s in persons if s['age']>18]
```

```
# -- !! Python code !! --
for s in l:
    print(s )
```

```
foreach( s : l )
    System.out.println (s )
```

Streams e listas

- ❖ The preferred method of iterating over a collection is to obtain a stream and perform aggregate operations on it.
- ❖ Aggregate operations are often used in conjunction with lambda expressions
 - to make programming more expressive, using less lines of code.
- ❖ `Package java.util.stream`
 - The key abstraction introduced in this package is stream.
- ❖ SEE MORE DETAILS DURING THE COURSE

Some examples using a list of strings

```
public static void listExample() {
    List<String> words = new ArrayList<String>();
    words.add("Prego");
    words.add("no");
    words.add("Prato");
    // old fashioned way to print the words
    for (int i = 0; i < words.size(); i++)
        System.out.print(words.get(i) + " ");
    System.out.println();

    // Java 5 introduced the foreach loop and Iterable<T> interface
    for (String s : words)
        System.out.print(s + " ");
    System.out.println();

    // Java 8 has a forEach method as part of the Iterable<T> interface
    // The expression is known as a "lambda" (an anonymous function)
    words.stream().forEach(n -> System.out.print(n + " "));
    System.out.println();

    // but in Java 8, why use a lambda when you can refer directly to the
    // appropriate function?
    words.stream().forEach(System.out::print);
    System.out.println();

    // Let's introduce a call on map to transform the data before it is printed
    words.stream().map(n -> n + " ").forEach(System.out::print);
    System.out.println();

    // obviously these chains of calls can get long, so the convention is
    // to split them across lines after the call on "stream":
    words.stream()
        .map(n -> n + " ")
        .forEach(System.out::print);
    System.out.println();
}
```



Prego no Prato
Prego no Prato
Prego no Prato
PregonoPrato
Prego no Prato
Prego no Prato

Some examples with an array of int

```
public static void arraysExample() {  
    int[] numbers = {3, -4, 8, 73, 507, 8, 14, 9, 3, 15, -7, 9, 3, -7, 15};  
  
    // want to know the sum of the numbers? It's now built in  
    int sum = Arrays.stream(numbers)  
        .sum();  
    System.out.println("sum = " + sum);  
  
    // how about the sum of the even numbers?  
    int sum2 = Arrays.stream(numbers)  
        .filter(i -> i % 2 == 0)  
        .sum();  
    System.out.println("sum of evens = " + sum2);  
  
    // how about the sum of the absolute value of the even numbers?  
    int sum3 = Arrays.stream(numbers)  
        .map(Math::abs)  
        .filter(i -> i % 2 == 0)  
        .sum();  
    System.out.println("sum of absolute value of evens = " + sum3);  
  
    // how about the same thing with no duplicates?  
    int sum4 = Arrays.stream(numbers)  
        .distinct()  
        .map(Math::abs)  
        .filter(i -> i % 2 == 0)  
        .sum();  
    System.out.println("sum of absolute value of distinct evens = " + sum4);  
}
```

```
sum = 649  
sum of evens = 26  
sum of absolute value of evens = 34  
sum of absolute value of distinct evens = 26
```

Parte 2: Persistência da Informação

Ficheiros

UA.DETI.POO

Introdução

- ❖ Sem capacidade de interagir com o "resto do mundo", o nosso programa torna-se inútil
 - Esta interação designa-se “input/output” (I/O)
- ❖ Problema → Complexidade
 - Diferentes e complexos dispositivos de I/O (ficheiros, consolas, canais de comunicação, ...)
 - Diferentes formatos de acesso (sequencial, aleatório, binário, caracteres, linha, palavras, ...)
- ❖ Necessidade → Abstração
 - Libertar o programador da necessidade de lidar com as especificidade e complexidade de cada I/O

Operações de entrada/saída (I/O)

Entrada

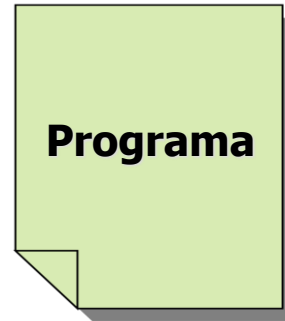
teclado



leitura



Programa



escrita



Saída

monitor



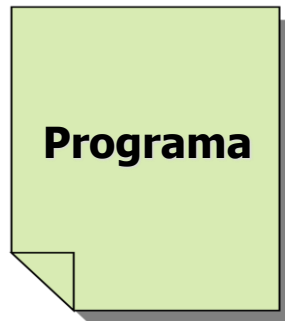
ficheiro



leitura



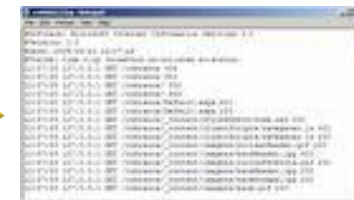
Programa



escrita



ficheiro



Java IO e NIO

- ❖ A linguagem java disponibiliza dois packages para permitir operações de entrada/saída de dados

- ❖ **Java IO**

- Stream oriented
- Blocking IO

- ❖ **Java NIO** (new IO)

- Buffer oriented
- Non blocking IO
- Channels
- Selectors



Ficheiros de Texto

Ficheiros – Classes principais

❖ Java IO

- File
- FileReader
- FileWriter
- RandomAccessFile

❖ Java NIO

- Path
- Paths
- Files
- SeekableByteChannel

java.io.File

- ❖ A classe *File* representa quer um nome de um ficheiro quer o conjunto de ficheiros num diretório
- ❖ Fornece informações e operações úteis sobre ficheiros e diretórios
 - `canRead`, `canWrite`, `exists`, `getName`, `isDirectory`, `isFile`, `listFiles`, `mkdir`, ...
- ❖ Exemplos:

```
File file1 = new File("io.txt");
```

```
File file2 = new File("C:/tmp/", "io.txt");
```

```
File file3 = new File("POO/Slides");
```



```
if (!file1.exists()) { /* do something */ }
```

```
if (!file3.isDirectory()) { /* do something */ }
```

Exemplo – Listar um Diretório

```
import java.io.*;

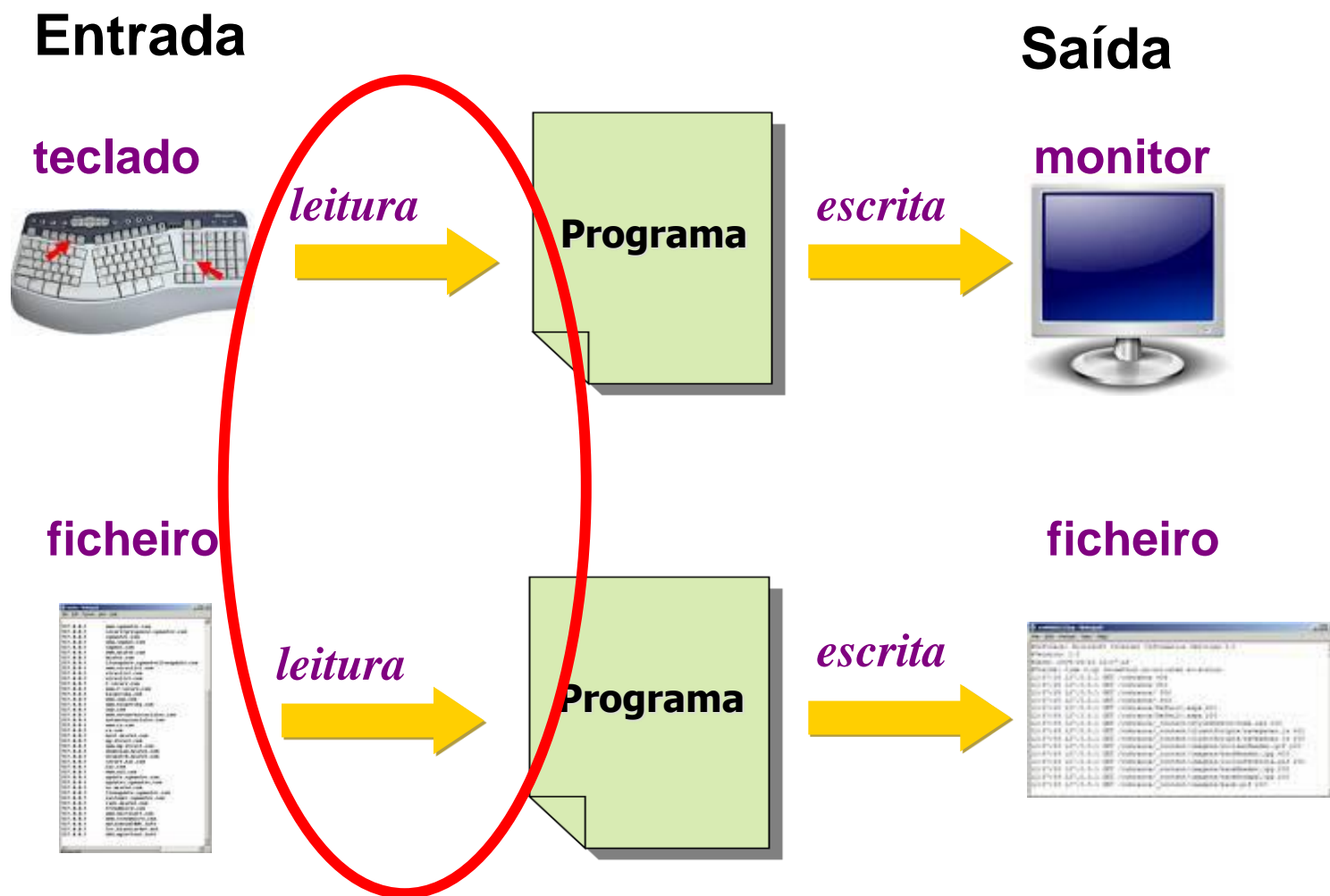
public class DirList {
    public static void main(String[] args) {
        File directorio = new File("src/");
        File[] arquivos = directorio.listFiles();
        for (File f : arquivos) {
            System.out.println(f.getAbsolutePath());
        }
    }
}
```

Com *java.nio*

Path dir = ...

```
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path entry: stream) { ... }
}
```

Operações de entrada/saída (I/O)



Ler dados ...

... usando **java.util.Scanner**



- ❖ Classe que facilita a leitura de tipos primitivos e de Strings a partir de uma fonte de entrada.

- Ler do teclado

```
Scanner sc1 = new Scanner(System.in);  
int i = sc1.nextInt();
```

- Ler de uma string

```
Scanner sc2 = new Scanner("really long\nString\n\t\tthat I want to pick apart\n");  
while (sc2.hasNextLine())  
    System.out.println(sc2.nextLine());
```

- Ler de um ficheiro

```
Scanner input = new Scanner(new File("words.txt"));  
while (input.hasNextLine())  
    System.out.println(input.nextLine());
```

Java obriga a identificar problemas

```
package aula01;
```

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

```
import java.io.File;
```

```
public class Ficheiro {
```

Run | Debug

```
public static void m
```

```
Scanner input = new Scanner(new File(pathname:"words.txt"));
```

```
while (input.hasNextLine())
```

```
System.out.println(input.nextLine());
```

```
input.close();
```

```
}
```

```
}
```

Unhandled exception type FileNotFoundException Java(16777384)

View Problem (Alt+F8) Quick Fix... (Ctrl+.) Fix using Copilot (Ctrl+I)

Java obriga a identificar problemas

```
package aula01;
```

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

```
import java.io.File;
```

```
public class Ficheiro {
```

```
1 package aula01;
```

```
2
```

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

```
4 import java.io.File;
```

```
5
```

```
6
```

```
7 public class Ficheiro {
```

```
8
```

```
Run | Debug
```

```
9 public static void m
```

```
10 Scanner input = new Scanner(new File(pathname: "w
```

```
11 while (input.hasNextLine())
```

```
12 | System.out.println(input.nextLine());
```

```
13 input.close();
```

```
14
```

```
15 }
```

```
16
```

```
17 }
```

VSCoide dá pistas do problema e propõe soluções

Unhandled exception type FileNotFoundException [Java(16777284)]

java.io.File.File(String pathname)

Creates a new File instance by converting the given pathname string into an abstract pathname. If the given string is the empty string, then the result is the empty abstract pathname.

• Parameters:

- **pathname** A pathname string

• Throws:

- **NullPointerException** - If the path

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

Quick Fix

- 💡 Add throws declaration
- 💡 Surround with try-with-resources
- 💡 Surround with try/catch
- 🌟 Fix using Copilot
- 🌟 Explain using Copilot

<https://www.geeksforgeeks.org/handle-an-ioexception-in-java/>

Java obriga a identificar problemas

```
package aula01;

import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class Ficheiro {
    Run | Debug
    public static void main(String[] args) {
        try (Scanner input = new Scanner(new File(pathname:"words.txt"))) {
            while (input.hasNextLine())
                System.out.println(input.nextLine());
            input.close();
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

... através de exceções: opções

❖ Necessário declarar que pode gerar exceção

```
public class TestReadFile {  
    public static void main(String[] args) throws FileNotFoundException {  
        Scanner input = new Scanner(new File("words.txt"));  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
    }  
}
```

❖ Ou lidar com exceção

```
public class TestReadFile {  
    public static void main(String[] args) {  
        try {  
            Scanner input = new Scanner(new File("words.txt"));  
            while (input.hasNextLine())  
                System.out.println(input.nextLine());  
            input.close();  
        } catch (FileNotFoundException e) {  
            System.out.println("Ficheiro não existente!");  
        }  
    }  
}
```



Problemas: passar para a frente

❖ Exemplo 1: sem tratamento de exceções

```
public class TestReadFile
{
    public static void main(String[] args) throws FileNotFoundException
    {
        Scanner input = new Scanner(new File("words.txt"));
        while (input.hasNextLine())
            System.out.println(input.nextLine());
    }
}
```

❖ O ficheiro "words.txt" deve estar:

- Na pasta local, se o programa for executado através de linha de commando
- Na pasta do projeto, caso seja executado a partir do IDE

Verificar e agir quando temos problemas

❖ Exemplo 2: try .. catch

```
public static void main(String[] args) {  
    try {  
        Scanner input = new Scanner(new File("words.txt"));  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
        input.close();  
    } catch (FileNotFoundException e) {  
        System.out.println("Ficheiro não existente!");  
    }  
}
```

– OU

```
public static void main(String[] args) {  
    Scanner input = null;  
    try {  
        input = new Scanner(new File("words.txt"));  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
    } catch (FileNotFoundException e) {  
        System.out.println("Ficheiro não existente!");  
    } finally {  
        if (input != null) input.close();  
    }  
}
```

Leitura de ficheiros de texto

❖ Exemplo 3: try-with-resources

- O código que declara e cria recursos é colocado na entrada try().
- Recursos são objetos que implementam AutoCloseable e que têm de ser fechados depois de usados.

```
public static void main(String[] args) {  
    try ( Scanner input = new Scanner(new File("words.txt"))) {  
        while (input.hasNextLine())  
            System.out.println(input.nextLine());  
    } catch (FileNotFoundException e) {  
        System.out.println("Ficheiro não existente!");  
    }  
}
```

java.nio – Leitura de ficheiros de texto

❖ Podemos usar métodos estáticos das classes **Files** e **Paths** do package `java.nio.file`.

❖ Exemplo 4:

```
public class ReadFileIntoList {  
    public static void main(String[] args) throws IOException {  
        List<String> lines = Files.readAllLines(Paths.get("words.txt"));  
        for (String ln : lines)  
            System.out.println(ln);  
    }  
}
```



Escrita de dados

Operações de entrada/saída (I/O)

Entrada

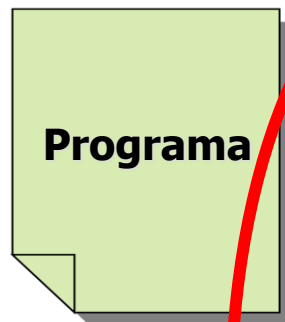
teclado



leitura



Programa



escrita



Saída

monitor



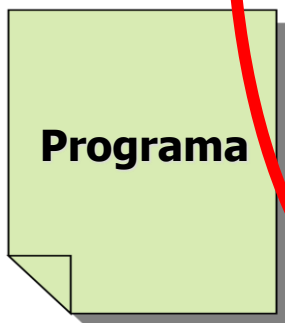
ficheiro



leitura



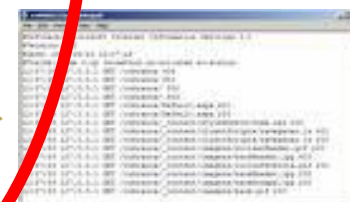
Programa



escrita



ficheiro



Escrita de ficheiros de texto

❖ classe java.io.PrintWriter

- Permite-nos usar os métodos println e printf para escrever em ficheiros de texto.
- Formata os valores de tipos primitivos em texto, tal como quando impressos no écran.

```
public class FileWritingDemo {  
    public static void main(String[] args) throws IOException {  
        PrintWriter out = new PrintWriter(new File("file1.txt"));  
        out.println("Fim de semana na praia");  
        out.printf("Viagem: %d\nHotel: %d\n", 345, 1000);  
        out.close();  
    }  
}
```

Escrita de ficheiros de texto – append

- ❖ Podemos acrescentar (append) mais informação a um ficheiro existente

```
public class FileWritingDemo {  
    public static void main(String[] args) throws IOException {  
        FileWriter fileWriter = new FileWriter("file1.txt", true);  
        PrintWriter printWriter = new PrintWriter(fileWriter);  
        printWriter.append("a acrescentar mais umas notas...\n");  
        printWriter.close();  
    }  
}
```

Sumário (Parte 2)

- ❖ java.io e java.nio
- ❖ Representar ficheiros e directórios com **File**
- ❖ Ler ficheiros de texto com **Scanner**
- ❖ Escrever ficheiros de texto com **PrintWriter**
- ❖ Muitas outras classes existem para manipular I/O
 - <https://docs.oracle.com/javase/tutorial/essential/io/>