

Aula 2 – Introdução a Java

Algoritmos e Estruturas de Dados

Introdução a Java

Continuação

Coletor de Lixo

Garbage Collector

Coletor de Lixo (Garbage Collector)

- Processo que se ativa de vez em quando na JVM para libertar memória do Heap
 - Quando a carga da CPU está baixa*
 - Quando o Heap começa a ficar cheio*
- Quando um objeto no Heap deixa de ser referenciado (ou seja não existem ponteiros para ele), é considerado como inacessível

```
ContaBancaria conta1 = new ContaBancaria(2923948);  
.  
.  
.  
conta1 = null;
```

Objecto criado no Heap

Objeto deixa de ser referenciado, nunca mais conseguiremos acede-lo

Coletor de Lixo (Garbage Collector)

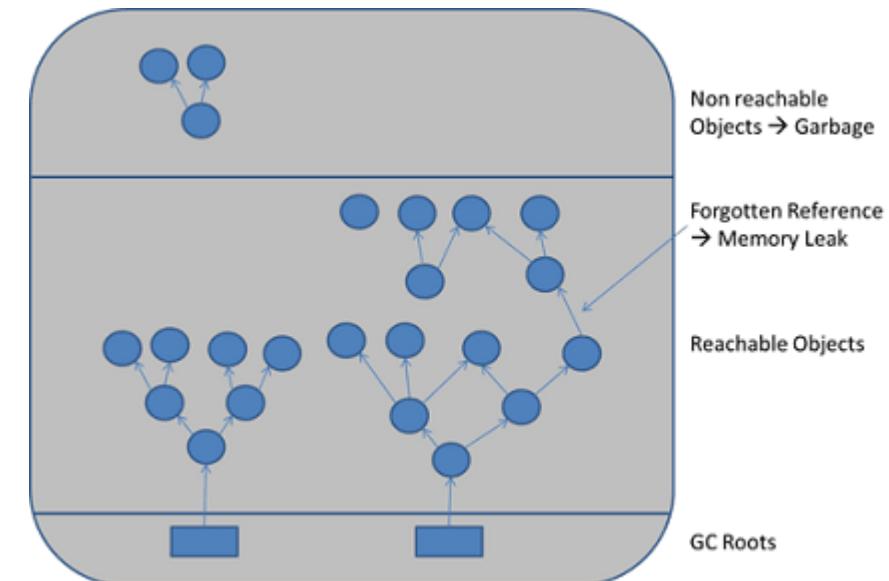
- Funcionamento do Coletor de Lixo
 - Quando ativado, são processadas 2 fases
 - Fase 1 – Marcação (Mark)
 - Objetos são percorridos a partir de umas “GC roots” especiais*
 - Objetos acedidos são marcados como “vivos”*
 - Objetos inacessíveis são considerados como lixo*

Objetos são percorridos a partir de umas “GC roots” especiais

Objetos acedidos são marcados como “vivos”

Objetos inacessíveis são considerados como lixo

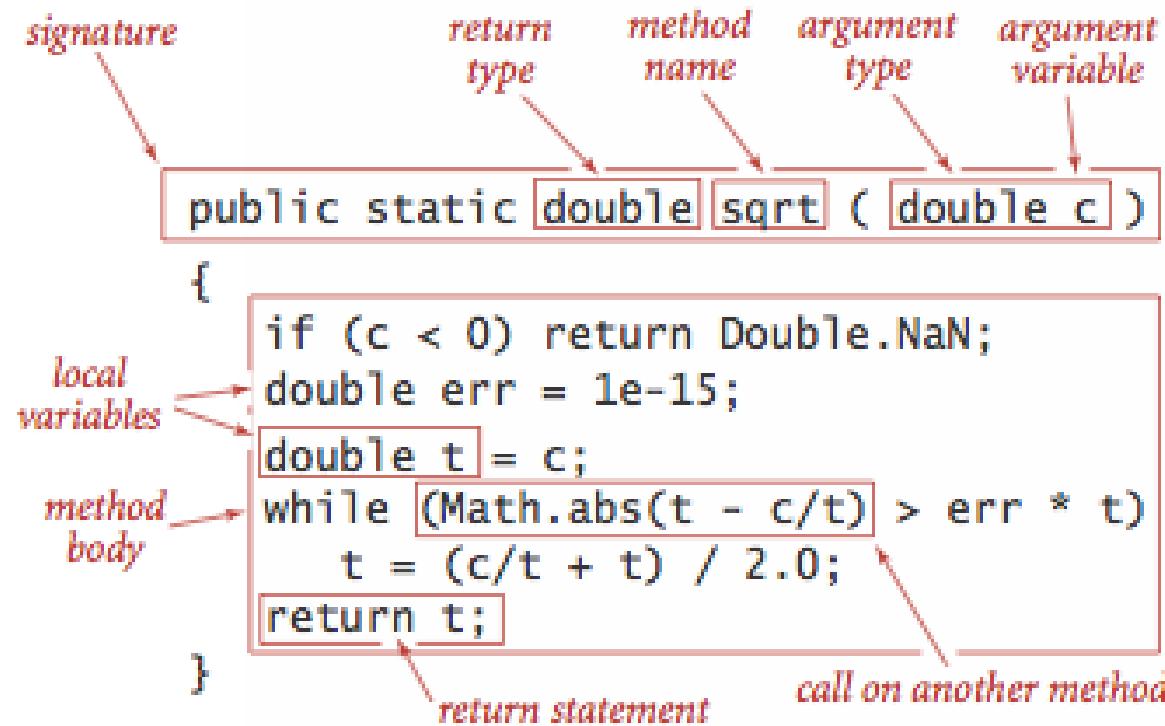
- Fase 2 – Varrimento (Sweep)
 - Memória ocupada pelos objetos inacessíveis é libertada*



Métodos estáticos

Métodos estáticos

- Métodos que não precisam de ser associados a uma instânciade uma classe para serem invocados



Exemplo euclides

- Método de cálculo para determinar máximo divisor comum entre dois inteiros
- Não é o + eficiente, mas é simples!

```
public static int euclides(int x, int y)
{
    while(x != y)
    {
        if(x < y) y -= x;
        else x -= y;
    }
    return x;
}
```

x	y
3	11
3	8
3	5
3	2
1	2
1	1

- O caso mais simples de um programa *Java* é uma biblioteca de métodos estáticos, com uma classe um método estático chamado *main*

```
public class Main {  
    public static int euclides(int x, int y)  
    {  
        while(x != y)  
        {  
            if(x < y) y -= x;  
            else x -= y;  
        }  
        return x;  
    }  
  
    public static void main(String[] args)  
    {  
        int mdc;  
        mdc = euclides(Integer.parseInt(args[0]), Integer.parseInt(args[1]));  
        System.out.println(mdc);  
    }  
}
```

argumentos recebidos quando o programa é executado



Classes e Interfaces

Classe Object

- Em Java todas as classes são subclasses de *Object*, herdando os seguintes métodos

retorno	assinatura	descrição
Class	<code>getClass ()</code>	devolve a classe do objecto
String	<code>toString ()</code>	retorna representação em String do objecto
boolean	<code>equals (Object that)</code>	recebe um objecto <i>that</i> e verifica se <i>that</i> é igual ao próprio objecto
int	<code>hashCode ()</code>	retorna um código de hash para ser usado numa tabela de dispersão

Classes e Interfaces

- Interface - define a assinatura de um conjunto de métodos
 - não pode ter membros
 - não pode especificar o corpo dos métodos
 - métodos têm que ser implementados por classes que “herdam” de uma interface

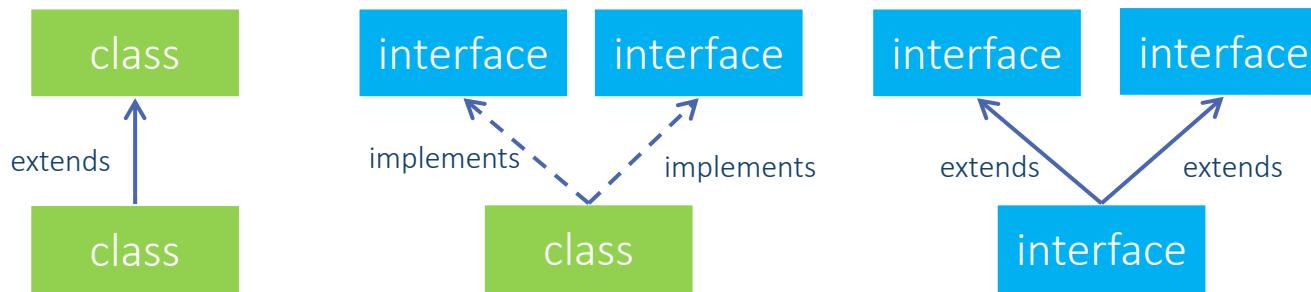
```
public interface MyInterface {  
    public void method1();  
    public void method2();  
}  
  
class A implements MyInterface  
{  
    public void method1()  
    {  
        System.out.println("Hello");  
    }  
    public void method2()  
    {  
        System.out.println("World");  
    }  
}
```

Classe A é obrigada a implementar todos os métodos da interface

Interfaces são um mecanismo interessante para garantir abstração de dados!

Classes e Interfaces

- Java
 - herança múltipla para interfaces
 - herança única para classes



Exemplos de Interfaces Java

- Interface Iterator e Iterable
 - especificam que uma coleção pode ser iterada
acedendo de forma sequencial a cada um dos elementos

Interface	Métodos especificados
java.lang.Comparable	compareTo()
java.util.Comparator	Compare()
java.lang.Iterable	Iterator()
java.util.Iterator	hasNext() next() Remove()

- for-each – ciclo desenhado para percorrer todos os elementos de uma coleção

```
List<String> palavras = Arrays.asList("lista", "de", "palavras");
```

```
for(int i = 0; i < palavras.size(); i++)
{
    System.out.println(palavras.get(i));
}
```

for tradicional

```
for(String p : palavras)
{
    System.out.println(p);
}
```

for-each

- for-each – ciclo desenhado para percorrer todos os elementos de uma coleção

```
for (String p : palavras)
{
    System.out.println(p);
}
```

- Implementado pelo java com base na interface Iterator

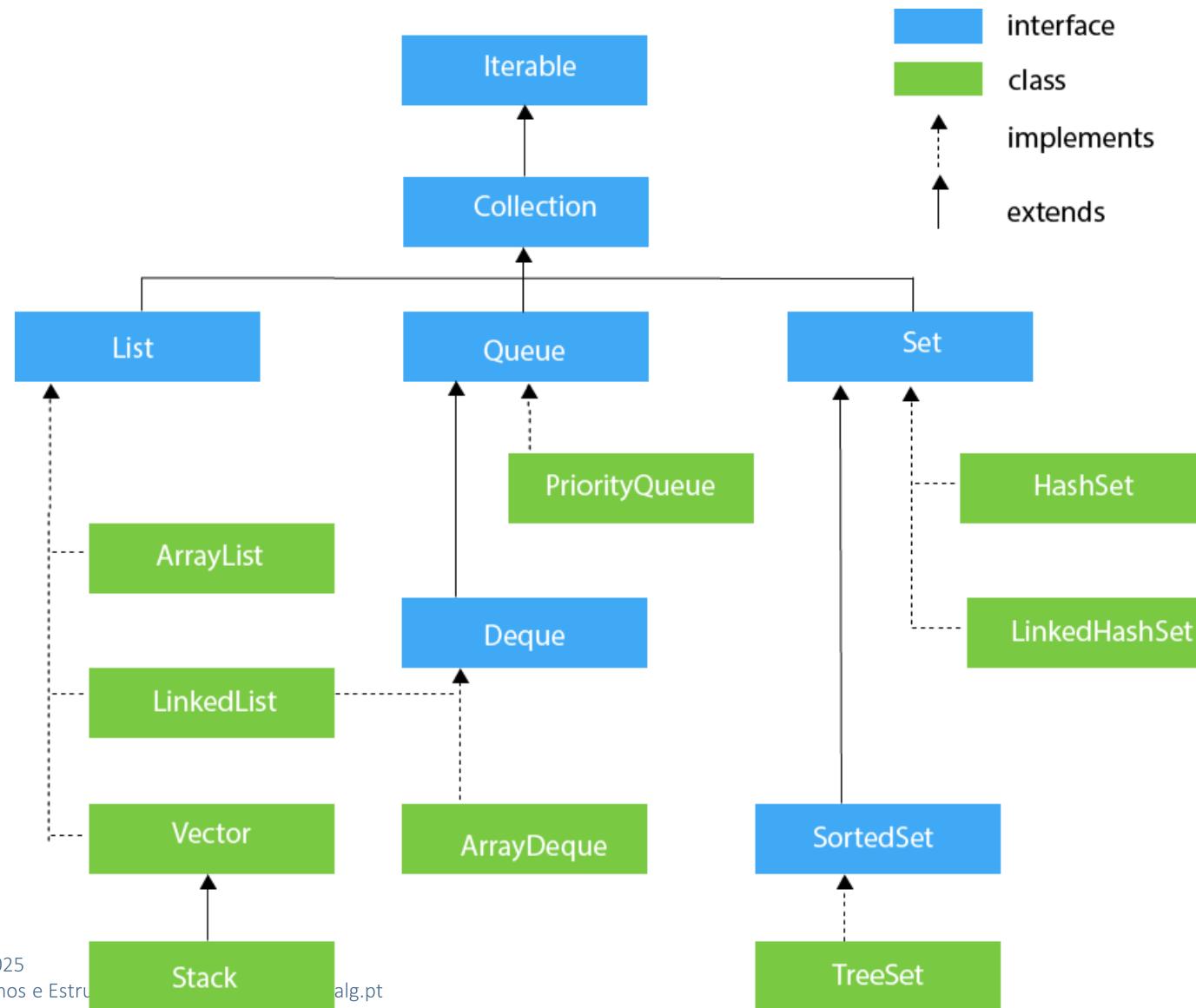
```
Iterator<String> it = palavras.iterator();
String p;
while (it.hasNext())
{
    p = it.next();
    System.out.println(p);
}
```

açúcar sintático
para



(quando vai ser
compilado o
código é
transformado
noutro)

Coleções em Java



Operador de Igualdade

Operador de Igualdade ==

- Operador ==
 - *Pode ser usado para comparar tipos primitivos*

```
int x = 3;  
int y = 4;
```

```
System.out.println(x == y);  
System.out.println(x == 3);
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe"  
false  
true
```

Operador de Igualdade ==

- Operador ==
 - *No entanto, temos de ter cuidado quando usamos == para comparar instâncias de objetos*

```
String a = new String("olá");
String b = new String("olá");
```

```
System.out.println(a == b);
System.out.println(a == "olá");
System.out.println(a == a);
```

Operador de Igualdade ==

- Operador ==
 - *No entanto, temos de ter cuidado quando usamos == para comparar instâncias de objetos*

```
String a = new String("olá");
String b = new String("olá");
```

```
System.out.println(a == b);
System.out.println(a == "olá");
System.out.println(a == a);
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe"
false
false
true

Process finished with exit code 0
```

Operador de Igualdade ==

- Operador ==
 - Quando aplicado a 2 objetos, este operador verifica apenas se são o mesmo objeto

```
String a = new String("olá");
String b = new String("olá");

System.out.println(a == b);
System.out.println(a == "olá");
System.out.println(a == a);
```

Observação: Este operador não compara o conteúdo de objetos.

Este é um erro de principiante muito comum em java.

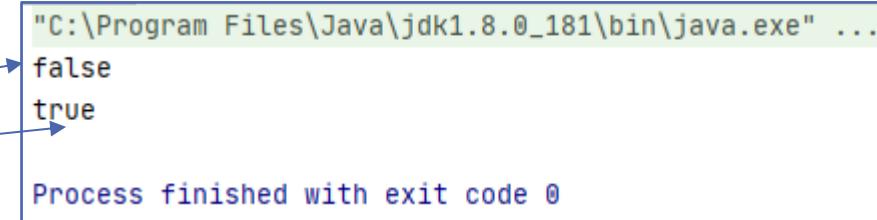
```
"C:\Program Files\Java\jdk-17\bin\java.exe"
false
false
true

Process finished with exit code 0
```

Método de igualdade *equals*

- A classe Object do Java implementa um método *equals*
 - Todas as classes do JRE têm um método *equals*
 - Este método deve ser usado para comparar objetos

```
public static void main(String[] args) {  
  
    String s1 = new String("ola");  
    String s2 = new String("ola");  
  
    System.out.println(s1 == s2);  
    System.out.println(s1.equals(s2));  
}
```



```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...  
false  
true  
Process finished with exit code 0
```

Método de igualdade *equals*

- Quando uma nova classe é criada por um programador
 - Vai herdar automaticamente o método equals da classe Object
 - Não é obrigatória a implementação do método equals...
 - Mas este método herdado tem o mesmo problema de ==

```
public boolean equals(Object obj) {  
    return (this == obj);  
}
```

Esta é a implementação do método equals da classe Object, e que é herdado por omissão por todas as classes

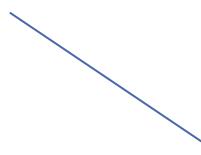
Método de igualdade *equals*

```
ContaBancaria c1 = new ContaBancaria(12344);
ContaBancaria c2 = new ContaBancaria(12344);

System.out.println(c1 == c2);
System.out.println(c1.equals(c2));
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe"
false
false

Process finished with exit code 0
```



A utilização do método *equals* não funciona como gostaríamos porque este método não foi redefinido

Redefinição de método equals

```
public boolean equals(ContaBancaria conta)
{
    if(conta == null) return false;
    return this.nib == conta.nib;
}
```

Método definido na classe ContaBancaria

Método usado para comparação quando o objeto a comparar é uma conta bancária

Consideramos que duas contas correspondem à mesma conta se o NIB for o mesmo

Redefinição de método equals

```
public boolean equals(ContaBancaria conta)
{
    if(conta == null) return false;
    return this.nib == conta.nib;
}
```

```
public boolean equals(Object o)
{
    if(o == null) return false;
    if(!(o instanceof ContaBancaria)) return false;

    return equals((ContaBancaria) o);
}
```

Método usado para comparação quando o objeto a comparar é de qualquer tipo

O operador *instanceof* verifica se um objeto é de uma determinada classe

Caso o objeto o seja uma Conta Bancária, fazemos uma conversão de tipo (*cast*) e simplesmente chamamos o método *equals* acima

Exceções

Exceções

- Em Java, existe um mecanismo elegante para lidar com algum tipo de problemas
- Exceção
 - Evento não desejado que pode ocorrer durante a execução de um programa que扰urba o normal funcionamento de um programa
 - Em Java, caso o programador o deseje, é possível tentar apanhar e tratar exceções

Tipos de exceções

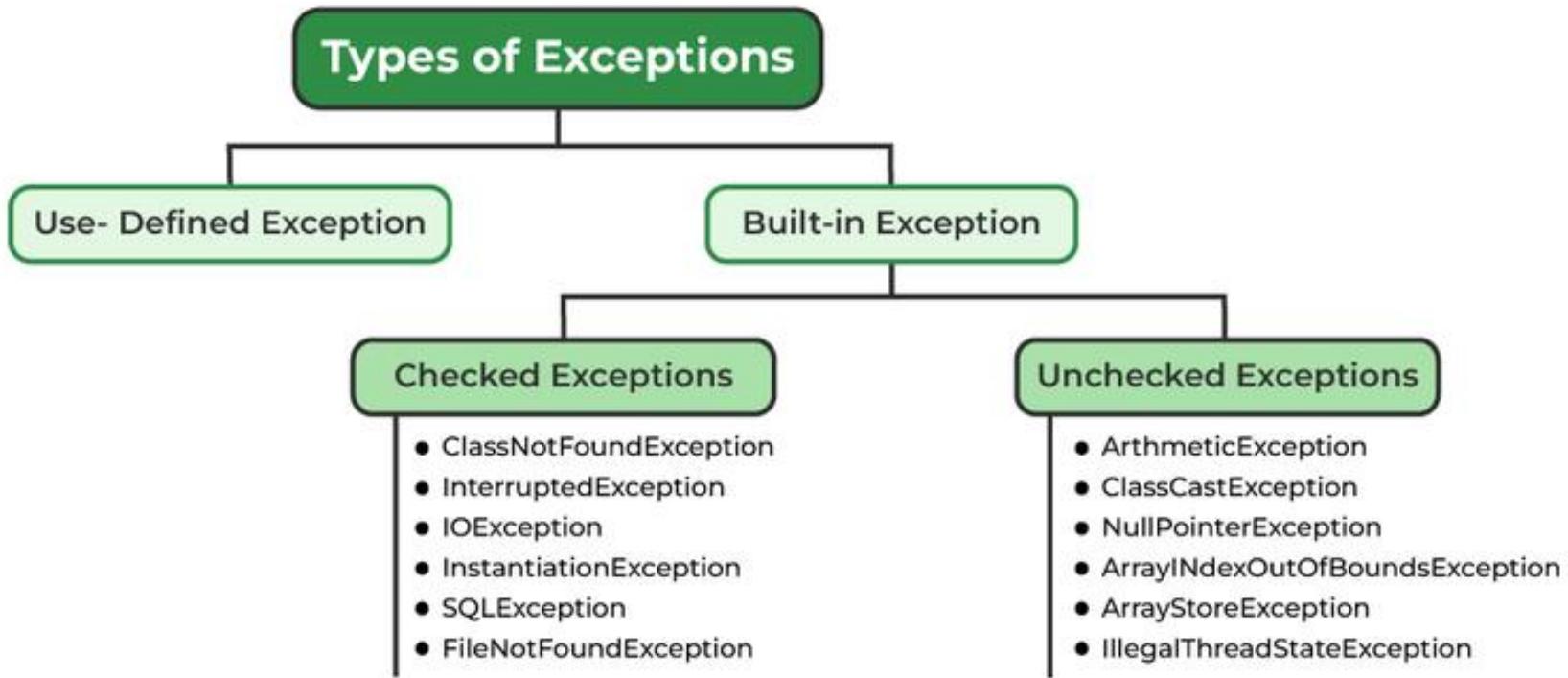
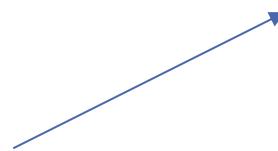


Imagen retirada de <https://www.geeksforgeeks.org/exceptions-in-java/>

Tratamento de Exceções

```
try
{
    ...
}
catch (IOException e)
{
    ...
}
catch (Exception e)
{
    e.printStackTrace();
}
```



Bloco try/catch

Usado quando sabemos que existem algumas exceções que podem ocorrer num bloco de código.

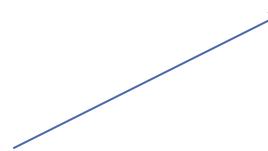
Colocamos as instruções potencialmente problemáticas dentro do bloco inicial do *try*

1) O Java vai tentar executar todas as instruções dentro do bloco try

Se tudo funcionar bem, não há qualquer problema, e o Java continua para a próxima instrução a seguir ao try/catch

Tratamento de Exceções

```
try
{
    ...
    ...
}
catch (IOException e)
{
    ...
}
catch (Exception e)
{
    e.printStackTrace();
}
```

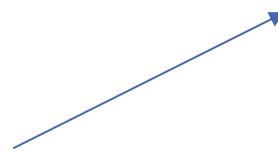


Bloco try/catch

Colocamos as instruções potencialmente problemáticas dentro do bloco inicial do *try*

2) Se alguma exceção ocorrer durante a execução do try, o Java para imediatamente a execução do bloco try (as restantes instruções não são executadas) e lança um objeto com informação sobre o que correu mal

```
try
{
    ...
    ...
}
catch (IOException e)
{
    ...
}
catch (Exception e)
{
    e.printStackTrace();
}
```



Bloco try/catch

3) Quando uma exceção é lançada dentro do corpo do try, o Java vai executar o bloco catch mais apropriado, caso exista.

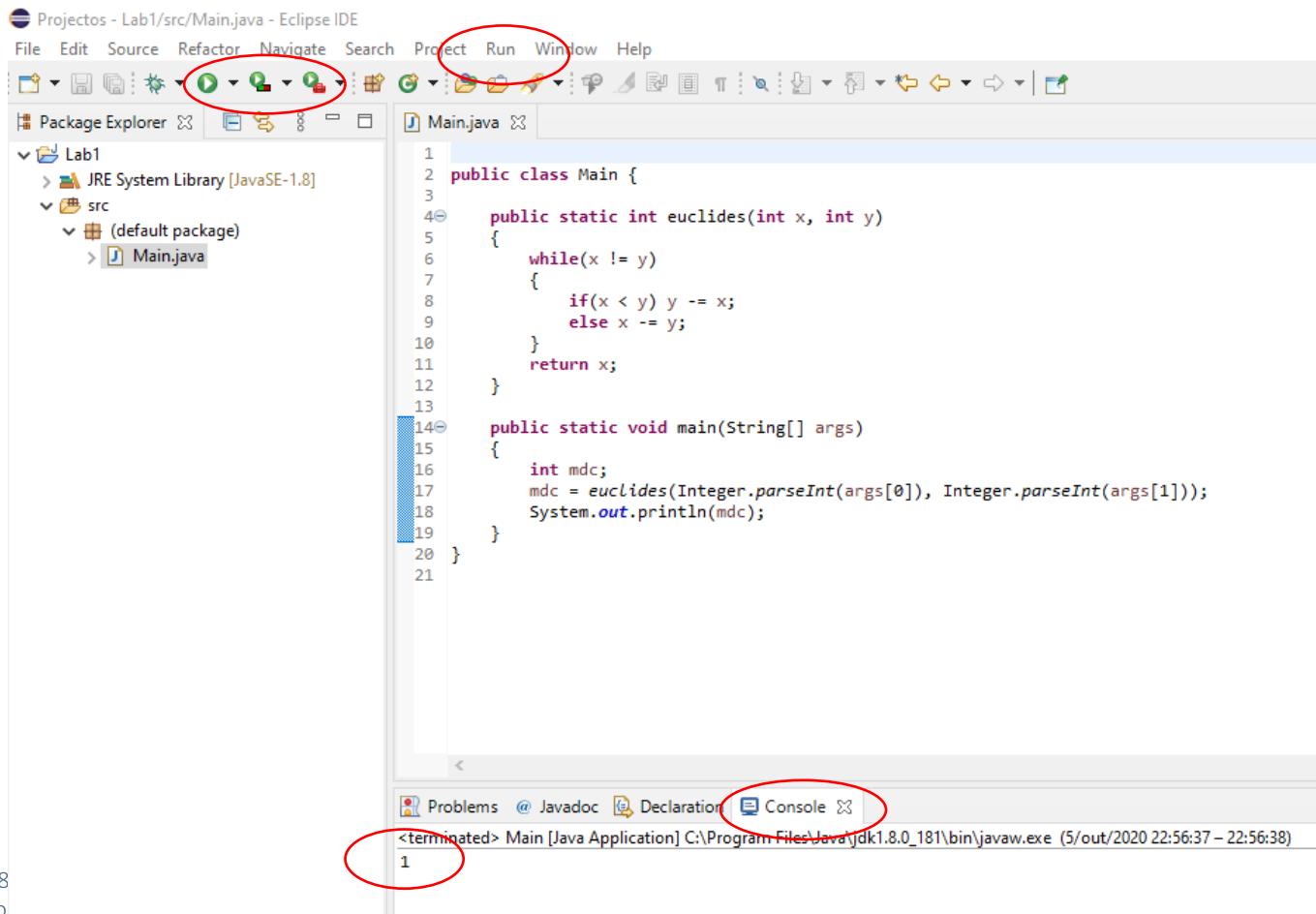
Neste exemplo, se a exceção lançada fosse do tipo IOException, iria executar o 1.º catch

O último catch é do tipo de exceção mais genérico, e por isso qualquer outra exceção virá aqui parar

Compilação e execução

Compilação e execução de um programa Java

- Eclipse/IntelliJ – desde que configurado, compila e executa por nós



The screenshot shows the Eclipse IDE interface with the following elements:

- Toolbar:** Circled in red, it includes icons for New Project, Open Project, Save, Run, Stop, and Exit.
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Package Explorer:** Shows a project named "Lab1" with a "src" folder containing a "Main.java" file.
- Main.java Editor:** Displays the following Java code:

```
1 public class Main {  
2     public static int euclides(int x, int y)  
3     {  
4         while(x != y)  
5         {  
6             if(x < y) y -= x;  
7             else x -= y;  
8         }  
9         return x;  
10    }  
11  
12    public static void main(String[] args)  
13    {  
14        int mdc;  
15        mdc = euclides(Integer.parseInt(args[0]), Integer.parseInt(args[1]));  
16        System.out.println(mdc);  
17    }  
18}  
19  
20}
```
- Console View:** Circled in red, it shows the output of the program execution:

```
<terminated> Main [Java Application] C:\Program Files\Java\jdk1.8.0_181\bin\javaw.exe (5/out/2020 22:56:37 - 22:56:38)  
1
```

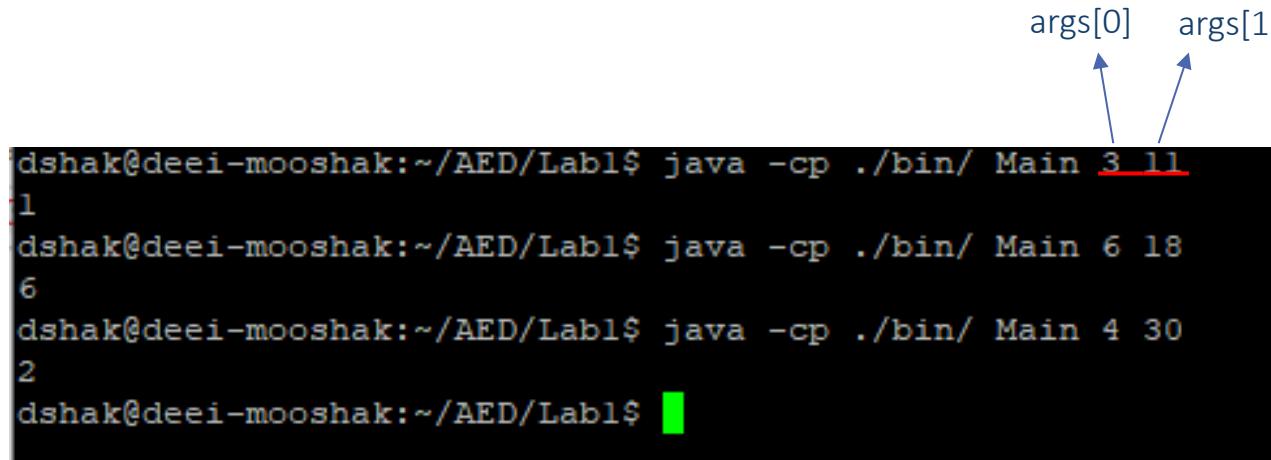
Compilação e execução de um programa Java

- Linha de comando
 - 1) Compilação – javac
 - d* directória onde colocar ficheiros compilados *.class*
 - cp classpath* onde procurar por classes necessárias para compilação
 - src/Main.java* ficheiro a compilar

```
dshak@deei-mooshak:~/AED/Lab1$ ls
bin  src
dshak@deei-mooshak:~/AED/Lab1$ javac -d bin -cp ./src/ src/Main.java
dshak@deei-mooshak:~/AED/Lab1$ ls bin
Main.class
dshak@deei-mooshak:~/AED/Lab1$ █
```

Compilação e execução de um programa Java

- Linha de comando
 - 2) execução - java
 - cp *classpath* onde procurar por classes necessárias para execução
 - Main* – classe com o método main a executar



The screenshot shows a terminal window with the following session:

```
dshak@deei-mooshak:~/AED/Lab1$ java -cp ./bin/ Main 3 11
1
dshak@deei-mooshak:~/AED/Lab1$ java -cp ./bin/ Main 6 18
6
dshak@deei-mooshak:~/AED/Lab1$ java -cp ./bin/ Main 4 30
2
dshak@deei-mooshak:~/AED/Lab1$ █
```

Annotations above the terminal output point to the command-line arguments: "args[0]" points to the value "3" and "args[1]" points to the value "11".