

Experiências nas aulas práticas

- aplicações em Java (na maioria...)

Ferramentas para desenvolvimento em Java

- Java Development Kit (compilador + interpretador)
- Java API: ([API Doc](#), o IDE também ajuda)
- utilitário para gestão das dependências do projeto (make; ant; maven; gradle)
- IDE (NetBeans, Eclipse, IntelliJ IDEA), que geralmente abrange os três anteriores

Em Sistemas Distribuídos, há muita comunicação entre aplicações.

Em geral, vamos precisar saber como executar uma aplicação sem o IDE... é preciso conhecer as dependências da aplicação e como iniciar a execução!

- não podemos depender de um IDE específico

Ainda o Java

- saber a versão de java, na linha de comandos:

\$ java -version

\$ javac -J-version

- verificar a versão (e outras propriedades), programaticamente:

```
System.out.println("System java version: "+System.getProperty("java.version"));
System.out.println("System java vendor: "+System.getProperty("java.vendor"));
System.out.println("System classpath: "+System.getProperty("java.class.path"));
```

Estas informações são úteis para despistar dúvidas sobre problemas relacionados com diferenças de versão ou ambiente, entre a execução num IDE e a execução em linha de comandos... Mesmo que estejamos a falar da mesma máquina.

outras propriedades

Qual é o separador de nomes de ficheiros no seu PC?

Qual é o encoding (UTF8, ISO-8859-15?) de ficheiros no seu PC? (file.encoding)

Qual é o nome de utilizador que executa a aplicação?

Exemplo de comando para executar:

\$ java -classpath build/classes package.Aula1 um dois tres

(para um projeto de outro IDE, terá de atualizar a pasta da classpath)

Note:

uma máquina pode ter vários ambientes (interpretador, compilador) de java. Às vezes há desencontros entre a versão de Java de um IDE e o Java que usa num servidor aplicacional ou noutra IDE. Não significa que haja um problema, mas deve estar consciente disso e saber como alterar.

Linguagem JAVA - Uma Revisão

Classes:

```
[public] [abstract: classe não instanciável (excepto pelas subclasses)]  
[final: não extensível] class NOME [extends SuperClass]  
[implements Interfaces] {  
  
    // class body: variáveis, construtores, métodos  
  
}
```

Variáveis de Classe e de Instância:

[Visibilidade: public, private, protected] [static: de classe] [final: imutável ou constante] tipo nome;

Tipos Primitivos:

char, byte, int(4 bytes), long(8), float, double, short, boolean

Tipos Referenciados: (objetos e arrays)

a variável representa um ENDEREÇO para um VALOR ou Grupo de Valores

o último caso representa um Array

- o Java não permite o uso explícito de endereços, como em C,
mas usa-se implicitamente, nas variáveis associadas a tipos referenciados.

Métodos:

```
[Visibilidade] [static: de classe]  
[abstract: não é implementado aqui]  
[final: não extensível] returnType name(argList)  
[throws Exception] {  
  
    // method body  
  
}
```

Construtores

Método especial para criar instâncias de uma classe

```
[AccessLevel] ClassName([argList]) {  
    [super(args)]  
  
    // body, usual with initializations  
}
```

Sobre Variáveis, Referências e Objetos em Java

- [factos importantes sobre objetos e referências](#)

Dúvidas comuns

1- Ler input da consola

versão simples (sem classes de apoio de alto nível)

```
r= System.in.read(bts);
s= new String(bts);
System.out.println("lido: "+r);
System.out.println("s: "+s+"\n\n");
```

Com a ajuda de métodos que escondem os detalhes de baixo nível, podemos ter algo como:

```
Scanner sc = new Scanner(System.in);
String s = sc.next();
// int i = sc.nextInt();
```

... mas deve conseguir tratar a questão a baixo nível, com operações sobre bytes.

2- Passar argumentos à aplicação, no momento da execução em linha de comandos:

```
$ java NomeDaClasse Argumento1 Argumento2...
```

A aplicação acede aos argumentos num array de String, o argumento do método de arranque da aplicação:

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

exercício

a) Implemente a classe Hello e verifique o acesso aos argumentos passados à aplicação.

```
public class Hello {
    public static void main(String[] args) {
        for (int i=0; i < args.length; i++) {
            System.out.println("Hello " + args[i]);
        }
    }
}
```

Teste a sua **execução**, tanto no IDE como na linha de comandos, passando diferentes argumentos.

b)

Independentemente da ferramenta que usa para editar (e executar), faça diretamente/manualmente a compilação da classe na linha de comandos... e volte a executar executar.

A intenção é perceber o que o IDE tem estado a fazer por si... e evitar dependência excessiva do mesmo.

Outras experiências elementares

Conversão de String para inteiro (analogamente para long, double)

(há outras formas... aqui usamos uma abordagem de baixo nível, com bytes)

```
byte[] b= new byte[128];
int lidos= System.in.read(b);
String s= new String(b, 0, lidos -1); // ou -2 no windows
System.out.println("lido: "+lidos);
System.out.println("s: "+s+"\n\n");
int valor= Integer.parseInt(s.substring(0,lidos-1));
System.out.println("valor: "+valor);
```

... poderia ter ocorrido um problema se a String fosse composta por caracteres no meio dos dígitos! (Tratamento de exceções!)

Informação sobre métodos, argumentos e classes existentes na linguagem:

API (é uma ferramenta para a aula!!!)

Vamos consultar na API:

- os métodos existentes na classe String
- classes de java.util
 - Hashtable, LinkedList, Date, **Calendar**, GregorianCalendar

exercício

Implemente uma aplicação em Java onde lê um valor da consola (sem Scanner), soma um e imprime o resultado...

exercício

Construa agora uma aplicação que recebe vários argumentos que representam nomes de pessoas. Os valores recebidos devem ser inseridos de forma ordenada num Vector ou numa [LinkedList](#) (ver API).

Implemente um método para efetuar o insertion sort. No final, o conjunto ordenado dos nomes deve ser mostrado na consola.

Exceções Java

Assinala-se uma situação anómala lançando uma exceção (tipo Exception)

Para quê? - para facilitar o tratamento da anomalia de acordo com o tipo específico de problema.

É bastante expressivo.

```
try {
    ...
    // linhas de código
    ...
}
catch (ExceptionType1 name1) { // so entra aqui com uma Exception deste tipo
    // o que fazer neste tipo de Exception
    statement(s)
}
catch (ExceptionType2 name2) { // so entra aqui com uma Exception deste tipo
    // o que fazer neste tipo de Exception
    statement(s)
} // pode haver OUTROS blocos CATCH, os mais genéricos ficam em baixo
[ // opcional
finally { // executa sempre esta parte em ultimo lugar, haja ou não Exception
    linhas de código
} ]
```

exercício

Altere o penúltimo exercício para que a ocorrência de caracteres num argumento onde se esperavam dígitos não provoque a paragem descontrolada do programa. Implemente o tratamento de exceções de modo a mostrar uma mensagem de aviso se algo de errado acontecer.

Interfaces

Uma interface define um protocolo para os objetos de uma classe, sendo útil para:

- representar um Tipo Abstrato de Dados
- declarar um conjunto de métodos que uma ou mais classes implementam, sem os detalhes específicos dessa implementação
- agrupar semelhanças entre classes que não tem de estar hierarquicamente relacionadas

Exemplo: `java.lang.Comparable`

exercício

Recorra à API de Java para descobrir as características um "objeto Comparable".

... isto é, um objecto compatível com o tipo Comparable

... ou um objecto que é instância de uma classe que implementa a interface Comparable

exercício

Implemente uma classe Classificacao que implemente a interface mencionada acima. Uma classificação deve ter um número de aluno, uma nota inteira, uma descrição (String) e observações (String).

Implemente outra classe NotasDeSO2 que representa um conjunto de classificações da disciplina. Esta classe deve possuir os métodos:

```
void adicionaClassif(Classificacao c)
```

```
List<Classificacao> getListaOrdenadaDeClassif() - este método devolve uma lista com as classificações ordenadas de modo decrescente. Se houver notas iguais o desempate é pelo nº de aluno.
```

Nota: é pouco provável que usemos nesta UC Módulos, Classes *Sealed* ou Records, mas fica a descrição.

Módulo

Um módulo é um bloco lógico, ou uma forma de arrumar pacotes e recursos (imagens, outros) relacionados entre si. Podemos expor alguns pacotes para acesso desde o exterior do módulo e reservar outros pacotes apenas para acesso interior ao módulo.

Records

especialmente úteis para o programador descrever dados (de estrutura simples), com sintaxe muito reduzida


Sealed Classes

forma de limitar a especialização da classe a um conjunto de subclasses específicas

- <https://www.baeldung.com/java-sealed-classes-interfaces>
- <https://www.baeldung.com/java-9-modularity>

Videos sobre execução em IDE

- [eclipse](#), [netbeans](#), [intelliJ-idea](#)
- VSCode: clique direito; executar Java; para alterar a versão, definir antes a variável JAVA_HOME (exemplo: \$ export JAVA_HOME=/usr/lib/jvm/jdk-17)

 [Contactar suporte do site](#) 

Nome de utilizador: [Rodrigo Alves](#) ([Sair](#))

[Resumo da retenção de dados](#)

[Obter a Aplicação móvel](#)

Fornecido por [Moodle](#)