# Aula 01
# Revisões
## Introdução ao LINUX; Revisões; Biblioteca IO do JAVA

*Programação II, 2016-2017*

*v3.4, 17-02-2017*

DETI, Universidade de Aveiro

01.1

**Objectivos:**
- Conhecimento mínimo sobre a utilização de sistemas operativos tipo UNIX;
- Revisões;
- Utilização da biblioteca *standard* do `Java` para entrada/saída.

## Conteúdo

**1 Introdução ao Sistema de Operação LINUX** **1**
1.1 Login . . . 2
1.2 Sistemas de ficheiros . . . 2
1.3 Linguagem de Comandos `bash` em UNIX . . . 3

**2 Programação: Revisões** **4**

**3 Ciclo da Programação** **5**
3.1 Comandos para Programação em JAVA . . . 6

**4 Introdução à Biblioteca de Entrada/Saída do JAVA** **6**

**5 Manipulação de ficheiros e directórios** **7**

**6 Escrita de ficheiros de texto em `Java`** **8**

**7 Leitura de ficheiros de texto em `Java`** **9**

01.2

# 1 Introdução ao Sistema de Operação LINUX

## Sistema de Operação

- Programa de *gestão do sistema computacional* (dos recursos de hardware e de software);
- Cria um *ambiente de interacção* entre os utilizadores e o computador;
- Fornece *protocolos de interacção* entre as aplicações e os recursos do sistema;
- Exemplos: LINUX, WINDOWS, MACOS.

01.3

## 1.1 Login

### Acesso ao LINUX nos Laboratórios

- O acesso de utilizadores ao sistema de operação impõe a sua *identificação*:

```
Login (a12345)
Password (******)
```

- Mesmo `login` que em Windows.

01.4

### Ambientes de Interacção

- Gráfico:
  - Organizado por janelas, ícones e menus de operações -> orientado à utilização pelo rato;
  - LINUX: GNOME, KDE.
- Terminal de texto:
  - Organizado por linguagens de comandos.
  - LINUX: `bash`, `csh`.

01.5

## 1.2 Sistemas de ficheiros

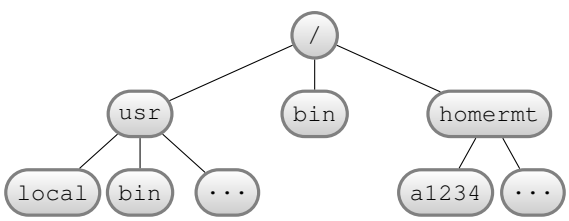
### Armazenamento de Informação: Ficheiros

- Elementos básicos:
  - *Nome*: identificador do ficheiro;
  - *Conteúdo*: informação a guardar e/ou aceder;
- O conteúdo dos ficheiros é organizado como uma sequência de bytes.
- A interpretação desses bytes depende dos programas que utilizam os ficheiros (pode ser texto legível ou informação binária).

01.6

### Sistema de Ficheiros

- Os ficheiros são organizados na memória de massa por um *sistema de ficheiros*;
- Os sistemas de ficheiros têm a si associados um conjunto de serviços (em cada SO) que permitem a sua utilização;
- Exemplos:
  - WINDOWS: FAT, NTFS;
  - LINUX: EXT2, EXT3.
- Os ficheiros são organizados por pastas (directórios) numa estrutura tipo árvore;
- Os ficheiros de informação constituem as folhas da árvore, e os directórios os seus ramos.




01.7

### Localização de Ficheiros

- O acesso a ficheiros é feito descrevendo o *caminho* que tem de ser percorrido na árvore de directórios;
- Essa descrição pode ser:
  - *Absoluta*:
    `/homermt/a12354/arca/aula01/Totoloto.java`
  - *Relativa*:
    `./arca/aula01/Totoloto.java`

01.8

### Acesso a ficheiros

- Um ficheiro (directório) pertence sempre a algum utilizador;
- Os *direitos de acesso* a ficheiros são geralmente classificados em três grupos:
  - Leitura;
  - Escrita;
  - Execução.
- Por sua vez esses direitos de acesso são aplicáveis a três tipos de entidades:
  - Utilizador;
  - Grupo;
  - Todos.

01.9

## 1.3 Linguagem de Comandos `bash` em UNIX

### Linguagem de Comandos `bash`

- A janela de terminal quando é executada, apresenta uma mensagem (*prompt*) que, geralmente, indica:
  - nome de utilizador;
  - identificação do computador (na rede local);
  - localização do directório de trabalho.
- Exemplo:
  `mos@butterfly:~/arca$`

01.10

### Comandos

- Um comando pode ser:
  - directamente implementado pela linguagem de comandos;
  - um ficheiro executável localizado algures no sistema de ficheiros.
- O comando é identificado e executado pelo seu nome.

01.11

### Exemplos de Comandos

- *whoami*: obtenção do nome do utilizador;
- *hostname*: obtenção do nome do sistema computacional;
- *cd* <DIRECTÓRIO>: mudança do directório de trabalho (dt);
- *pwd*: obtenção do encaminhamento absoluto do dt;
- *ls* (ou `ls -l`): listagem (detalhada) do conteúdo do dt;
- *cat* <FICHEIRO>: impressão do conteúdo de um ficheiro de texto;
- *mv* <NOME> <NOVO NOME>: alteração do nome de um directório/ficheiro;
- *mv* <NOME> <DIRECTÓRIO>: alteração da localização de um directório/ficheiro;
- *cp* <FICHEIRO> <NOVO NOME>: cópia de um ficheiro;
- *rm* <FICHEIRO>: remoção de um ficheiro;
- *mkdir* <DIRECTÓRIO>: criação de um directório;
- *rm -r* <DIRECTÓRIO>: remoção de um directório e do seu conteúdo;
- *man* <NOME-COMANDO>: página de manual de um comando.

01.12

### Composição de Comandos

- As linguagens de comandos têm mecanismos que permitem a *construção de comandos* mais complexos a partir de comandos mais simples:
  - *Shell scripts*.
- Permitem também a combinação de comandos:
  - *Piping*;
  - Exemplo:

```
cat Totoloto.java | more
```

01.13

## 2 Programação: Revisões

**Programação**: Disciplina de construção de programas, autónoma e automaticamente executáveis em computadores, visando a resolução de determinados problemas.

Na construção de programas o programador tem em conta dois aspectos distintos:

- *Registo de Informação*
  Forma fiável de registar informação, seja ela representada por números, texto ou outro qualquer tipo.
- *Algoritmo*
  Meio para se poder expressar acções que, fazendo uso da informação registada, possam gerar nova informação.

Ao algoritmo cabe assim a tarefa de decidir a sequência de acções a serem executadas pelo programa. Por sua vez, ao registo de informação cabe o papel de memorizar toda a informação desejada. O algoritmo frequentemente faz uso do estado actual da informação armazenada para decidir o caminho a percorrer.

Note que, muito embora esta distinção seja muito importante, por vezes qualquer um dos aspectos pode servir para um fim desejado. Por exemplo, o valor de $\pi$, tanto pode provir de uma função, com um algoritmo interno de cálculo, como de uma variável pré-definida que já tenha esse valor registado.

**Programa**: Em síntese, um programa será um algoritmo que recebendo informação, vai gerando nova informação, por forma a resolver o problema para o qual foi, ou está a ser, desenvolvido.

01.14

Em Programação 1 já vimos que o registo de informação faz uso de um mecanismo das linguagens de programação designado por *variável*. Os programas acedem a essa *variável* através de *identificadores* havendo *instruções* que permitem registar nova informação e *expressões* que permitem aceder à informação lá registada. A essas variáveis está associado um *tipo*, que define o tipo de informação que lá pode ser registada. Ao contrário do nosso cérebro, as linguagens de programação garantem a fiabilidade, consistência e determinismo no acesso e registo dessa informação. Assim, uma vez registado um determinado valor numa variável, mesmo que o programa demore um milénio a ser executado, acessos a essa variável devolverão sempre o último valor lá registado.

Vimos também que o processamento de informação – os *algoritmos* – são expressos, em linguagens imperativas como é o caso do `Java`, como uma sequência de instruções (ou comandos).[1]

As *instruções* mais importantes à disposição do programador são as seguintes:

- Atribuição de valor a variáveis (operadores =, +=, ...);
- Bloco de instruções `{...;...;...}`;
- Instruções condicionais (`if` e `switch`);
- Instruções repetitivas (`while`, `for`);

[1] Donde vem a designação de imperativo.

- Invocação de procedimentos e funções.

Pode-se demonstrar que em teoria, para além do registo de informação, bastam os quatro primeiros tipos de instrução para ser possível expressar algoritmos que resolvam qualquer problema computável. Na prática, no entanto, a complexidade inerente à programação faz com que sejam necessários outros mecanismos de *gestão de complexidade* (como é o caso dos procedimentos e das funções)

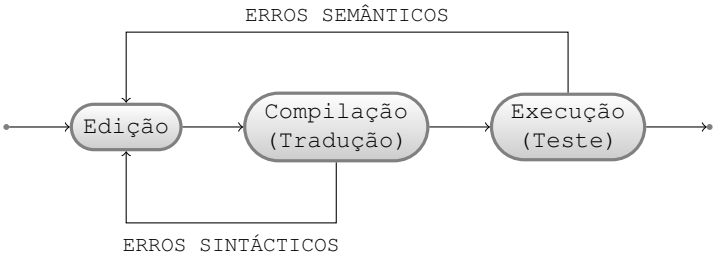
Os procedimentos e funções (métodos), são a concretização da chamada *abstracção algorítmica*. Os métodos permitem a criação, por parte do programador, de novos comandos ou novas funções[2] a serem (re)utilizados pelo programador, sem que para a sua utilização seja necessário conhecer e até por vezes compreender o algoritmo utilizado para os implementar. Dessa forma o esforço na construção de novos programas pode ser substancialmente reduzido.

Para além da abstracção algorítmica, os métodos permitiam criar novos *contextos de existência* de variáveis. Esses contextos de existência são circunscritos ao corpo dos próprios métodos[3]. Para esse fim podíamos declarar variáveis internas ao próprio método, assim como passar informação por intermédio de parâmetros (que internamente ao método funcionam como variáveis). Assim, os métodos funcionam como entidades com alguma autonomia, um pouco como se fossem "pequenos" programas, escondendo a existência de algumas variáveis, não havendo a necessidade dos actuais e futuros clientes dos mesmos saberem sequer da sua existência. 01.15

# 3 Ciclo da Programação

A *programação* – actividade de concepção e construção de programas – pode ser vista como sendo a implementação de uma espécie de *receita*, fazendo uso de instruções e variáveis, por forma a que o problema seja resolvido.

## Ciclo da Programação




01.16

O desenvolvimento de programas pode envolver dois tipos diferentes de erros: *sintácticos* e *semânticos*. Os primeiros são relativamente fáceis de detectar e corrigir. Indicam falhas elementares no uso da linguagem como, por exemplo, quando se omite o símbolo `;` no fim de cada instrução. Os segundos, são em geral mais difíceis de detectar e corrigir, e indicam as situações em que o programa não faz o que era suposto fazer[4]. Por exemplo a função seguinte (que era suposto devolver o número elevado ao quadrado), embora sintaticamente correcta, está semanticamente errada:

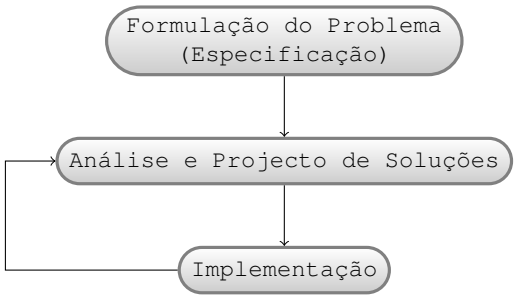
```
static int square(int n)
{
   return n*2; // devia ser n*n
}
```

[2] As funções diferem conceptualmente dos procedimentos por terem a si associadas um resultado. Podem ser encaradas tanto como alternativas ao registo de informação, como também a um procedimento que armazena um valor numa variável e depois o devolve como resultado.

[3] O mesmo acontece com os blocos (`{...}`).

[4] "Semântica" vem de "significado".

### Construção de Programas




01.17

Na construção de programas, o programador vê-se frequentemente envolvido num ciclo (que se deseja e espera não vicioso nem infinito), de concepção, implementação e depuração, até que o programa esteja correcto.

## 3.1 Comandos para Programação em JAVA

### Comandos para Programação em JAVA

- *Editores de texto*:
  - `geany Totoloto.java`
  - `gedit Totoloto.java`
  - `gvim Totoloto.java`
- *Compilador*:
  - `javac Totoloto.java`
- *Execução*:
  - `java Totoloto`

01.18

# 4 Introdução à Biblioteca de Entrada/Saída do JAVA

### Serviços de Escrita no Ecrã (saída)

- Acessível pela entidade: `System.out`
- Conjunto de serviços definido no módulo: `PrintStream`
  - Escrita através dos métodos: `print(...)`, `println(...)` e `printf(...)`
- Exemplo:

```
System.out.println("Boa tarde!");

double PI = 3.141596;
System.out.print("PI = " + PI);
System.out.println();
System.out.printf("2*2 = %d\n",2*2);
```

01.19

### Serviços de Escrita no Ecrã (saída)

- Exemplo 2:

```
import static java.lang.System.*;

(...)

  out.println("Boa tarde!");

  double PI = 3.141596;
  out.print("PI = " + PI);
  out.println();
  out.printf("2*2 = %d\n",2*2);
```

01.20

### Serviços de Leitura do Teclado (entrada)

- Acessível pela entidade: `System.in`
- Conjunto de serviços definido no módulo: `Scanner`
  - Leitura através dos métodos: `nextInt()`, `nextLine()`, etc.
- É necessário criar uma entidade de leitura do teclado:

```
import static java.lang.System.*;
import java.util.Scanner;
...
  final static Scanner in = new Scanner(System.in);
```

- Exemplo:

```
  out.print("Nome: ");
  String nome = in.nextLine();
  out.print("Idade: ");
  int idade = in.nextInt();
  in.nextLine(); // "consumir" restante linha
```

01.21

## 5 Manipulação de ficheiros e directórios

### Ficheiros e Directórios

- O que é um ficheiro?
  - Estrutura de armazenamento de informação;
  - Uma sequência de “0” e “1” armazenados (informação binária).
- O que é um directório?
  - Tipo especial de ficheiro que armazena uma lista de referências a ficheiros.
- Características:
  - Localização no sistema de ficheiros (directório e nome);
  - Têm a si associadas permissões de leitura, escrita e execução.

01.22

### Utilização de ficheiros em `Java`

- Tipo de dados `File` (`java.io.File`);
- Permite:
  - Confirmar a existência de ficheiros;
  - Verificar e modificar as permissões de ficheiros;
  - Verificar qual o tipo de ficheiro (directório, ficheiro normal, etc.);

  - Criar directórios;
  - Listar o conteúdo de directórios;
  - Apagar ficheiros.
  - ...
- Documentação: `view-javadoc File`
- Exemplo: comando `ls`

01.23

### Ficheiros de texto

- Os dados são interpretados e transformados de acordo com formatos de representação de texto;
- Cada carácter é codificado (`ASCII`, `Unicode`, `UTF-8`, ...)
- Comando (`linux`): `iconv`
- Comandos (`linux`, prog2): `latin1-2-utf8`, `utf8-2-latin1`

01.24

### Texto em `Java`

- Os tipos `char` e `String` codificam o texto com a codificação unicode 16 *bits*;
- Esse detalhe de implementação do `Java` é, no entanto, transparente para o programador;
- Os serviços de entrada/saída fazem automaticamente a tradução de ou para a codificação escolhida;
- Existem constantes literais para expressar caracteres latin1 (`'\xxx'`) ou unicode (`'\uxxxx'`);
- Existem também constantes literais para alguns caracteres especiais:
  - `'\n'`: nova linha;
  - `'\t'`: tabulação horizontal;
  - `'\"'`: carácter `"`
  - `'\''`: carácter `'`
  - `'\\'`: carácter `\`

01.25

## 6 Escrita de ficheiros de texto em `Java`

- Tipo de dados `PrintWriter` (`java.io.PrintWriter`);
- Interface similar à do `PrintStream` (`System.out`);
- Utilização:
  1. Criar uma entidade (objecto) `File` associada ao nome do ficheiro desejado:

```
File fout = new File(nomeFicheiro);
```

  2. Declaração e criação de um objecto tipo `PrintWriter` associado a esse objecto tipo File:

```
PrintWriter pwf = new PrintWriter(fout);
```

  3. Escrever sobre o ficheiro:

```
pwf.println(line); ...
```

  4. Fechar o ficheiro

```
pwf.close();
```

01.26

### Exemplo: escreve números primos

```
1. Ler nome do ficheiro
2. Ler número máximo: n
3. Abrir ficheiro para escrita
4. Percorrer em i todos os números de 2 a n
   4.1 Se i é primo então
      4.1.1 Escreve i no ficheiro
5. Fecha ficheiro
```

1. A abertura de um ficheiro para ser escrito é uma operação que pode falhar (por exemplo, quando não há permissão para escrever no directório onde pretendemos criar o ficheiro);
2. Nessa situação, a linguagem `Java` utiliza um mecanismo de *excepções*;
3. Este mecanismo será tópico de aulas futuras, pelo que para já deve utilizar a estratégia definida em Programação I para lidar com esta situação:
   (a) Em todos os métodos onde há operações de abertura de ficheiros, adicionar à assinatura do método a instrução: `throws IOException`

01.27

## 7 Leitura de ficheiros de texto em `Java`

- Tipo de dados `Scanner` (`java.util.Scanner`);
- Em vez do `System.in` associar o `Scanner` ao ficheiro a ler;
- Utilização:
  1. Criar uma entidade (objecto) `File` associada ao nome do ficheiro desejado:

```
File fin = new File(nomeFicheiro);
```

  2. Declaração e criação de um objecto tipo `Scanner` associado a esse objecto tipo `File`:

```
Scanner scf = new Scanner(fin);
```

  3. Ler do ficheiro:

```
while(scf.hasNextLine())
  String line = scf.nextLine();
```

  4. Fechar o ficheiro:

```
scf.close();
```

01.28

### Exemplo: comando `cat`

```
1. Ler nome do ficheiro
2. Abrir ficheiro para leitura
3. Enquanto não chegar ao fim do ficheiro
   3.1. Ler linha do ficheiro
   3.2. Escrever linha na consola
4. Fecha ficheiro
```

01.29

## Exemplo: comando `grep`

```
1. Ler nome dos ficheiros de entrada e saída
2. Ler texto do padrão
3. Abrir ficheiro para leitura: scf
4. Abrir ficheiro para escrita: pwf
5. Enquanto não chegar ao fim de scf
   5.1. Ler uma linha de texto de scf
   5.2. Se linha verifica padrão então
      5.2.1 Escrever linha em pwf
6. Fechar ficheiros
```

01.30