A linguagem JAVA

Programação na Web

Luís Manuel Borges Gouveia

Motivação

- início de 1997
  - mais de 50 milhões de utilizador da Internet
  - aumento de utilizadores a uma taxa de 100% ano
  - estimam-se mais de 200 milhões de utilizadores em 2002
  - em 2010 estima-se existirem mais ligações para uso da Internet do que linhas telefónicas!

Motivação

- na Internet?
  - mais precisamente na World Wide Web
  - introduzida no início dos anos 90
  - uma das histórias de sucesso das TI's
  - proporciona um meio de partilha de informação a baixo custo

Motivação

- qual a vantagem?
  - ao alcance de uma chamada telefónica local é possível “ler” texto e gráficos de qualquer parte do mundo
  - informação organizada em páginas, com um nome único, acessíveis globalmente
  - necessário um navegador, mas fácil de utilizar (hipertexto)

Motivação

- desvantagens?
  - é um pouco lento
  - infraestrutura sobrecarregada
  - reduzida interactividade
    - páginas “escritas” em HTML que permite leitura de conteudos multimédia, saltos para outras páginas, carregamento de gráficos
    - o JAVA estende estas capacidades possibilitando a criação de páginas interactivas...

Motivação

- porquê usar JAVA?
  - pela interactividade…
  - globalmente aceite como a linguagem de programação para a Web
  - tornou-se uma norma para a indústria de software
  - possui poucos concorrentes para o desenvolvimento de aplicações na Web

O que é o JAVA

- uma linguagem de programação
  - orientada a objectos
  - proveniente da Sun Microsystems
  - permite o desenvolvimento de dois tipos de programas
    - APPLETS, que se destinam a ser executados em conjunto com um navegador da web
    - Aplicações, que constituem os tradicionais programas (e sem recurso a um navegador para serem executados)

O que é o JAVA

- são os APPLETS que nos interessam
  - são multiplataforma
  - permitem o desenvolvimento de páginas interactivas para a web
    - jogos
    - formulários
    - visualizadores
    - demos
    - o que a imaginação poder e o trabalho concretizar!

Desenvolvimento de software

- enquanto o hardware comporta-se sempre de uma forma previsível (quando não há avaria!), o software já não…
- mesmo para a aplicação perfeita seria virtualmente impossivel de provar esse facto
- inumeras metodologias de projecto de programação foram desenvolvidas
  - programação orientada a objectos (OOP) é uma delas!

POO

- POO é uma das melhores formas de desenvolver software com o mínimo de erros
  - o JAVA é uma OOP…
  - deve-se tirar partido desse facto!
  - Necessário saber o que são:
    - objectos
    - diferença entre objectos e classes
    - atributos
    - herança e sua importância

O que são objectos

- mundo exterior como um conjunto de objectos
  - o carro , a casa, os animais, as plantas; num programa: os botões, as imagens, as caixas de textos,…
- a maioria dos objectos é feito de outros objectos
  - os automoveis são concebidos como agrupamentos de objectos cuja concepção está entregue a várias equipas
  - não existe necessidade de possuir o conhecimento de como determinado objecto está estruturado ou funciona para se poder combinar com outros

O que são objectos

- na POO, com objectos que se comportam de forma bem determinada, é possível realizar a sua combinação sem ser necessário saber o que acontece no seu interior
  - para um objecto que ordene números, só é necessário fornecer os números a esse objecto e esperar até que este retorne os números ordenados; não é preciso saber como estes são ser ordenados
- o ponto fulcral da POO é ver uma aplicação como um conjunto de objectos que interagem entre si
  - permite a concentração no projecto interno de um objecto sem a preocupação do efeito que este terá noutros objectos (o interface entre objectos tem de ser claramente definido)

Classes

- em POO, classes e objectos são coisas diferentes!
- no mundo real existem muitos objectos que são do mesmo tipo
  - a partilha de características é um modo útil de agrupar objectos
  - todos os livros no mundo são membros da mesma **classe** livro, que possui características ou **atributos**: os livros tem páginas, uma capa e contém palavras ou imagens
  - a Biblia é um **objecto** livro, que é um membro da **classe** livro.
  - em termos POO, diz-se que a Biblia é uma **instância** da **classe** livro

Classes

- uma linguagem OO permite criar classes e objectos que pertencem a essas classes (outro exemplo):
  - a classe construção…, existem muitas construções, mas só estamos numa, num dado momento
  - construção é a classe, isto é, o formulário para todas as construções, mas a construção onde nos encontramos é uma instância dessa classe; cada instância é um objecto
  - a classe não se refere a uma dada construção em particular; é apenas o formulário que descreve todas as construções. Uma instância da classe contrução é uma construção particular, é um objecto; uma instância de uma classe é um objecto
  - uma moradia, um prédio e uma igreja são diferentes, mas todos os objectos partilham um conjunto comum de atributos; portas, paredes e uso

Atributos e comportamento

- todas as instâncias de uma classe (os objectos dessa classe) partilham os mesmos atributos.
  - alguns dos atributos para a classe construção podem ser: o nº de pisos, o nº de janelas, a área coberta, o uso da construção
- os valores dos atributos são diferentes para diferentes construções e são designados por variáveis de instânciação
  - as variáveis podem permanecer constantes para a vida do objecto (o número de pisos da construção) ou podem mudar (o uso da construção)

Atributos e comportamento

- as classes definem conjuntos de operações que são feitas em cada um dos objectos para mudar o valor das variáveis ou proporcionar informação sobre estas
  - operação para modificar o uso da construção de trabalho para lazer; igualmente devem existir operações que reportem o seu estado
- as classes fornecem uma referência que define tanto as variáveis como as operações que podem ser realizadas; estas operações são designadas por métodos
  - após um a classe ser definida, os objectos pertencentes a essa classe (instâncias) podem ser criados
  - JAVA oferece várias classes para uso, constituindo a base de trabalho, mais elaborada, para esforços de programação

Herança

- é útil dividir uma classe em subclasses, para assim partilharem algumas características
  - a classe construção pode ser dividida em subclasses: habitação, escritório, loja. Construção é a superclasse das subclasses habitação, escritório, loja
- uma subclasse herda os atributos da superclasse e todos os métodos que operam essa superclasse
  - as subclasses habitação, escritório, loja partilham o atributo que define o uso da construção e os respectivos métodos para visualizar e alterar esse atributo

Herança

- as subclasses podem possuir atributos adicionais aos da superclasse a que pertencem
  - a subclasse habitação pode possuir um atributo que especifica o número máximo de pessoas que lá podem viver e alguns métodos associados
- as subclasses também podem alterar, para si próprias, métodos herdados das superclasses
  - a subclasse loja pode modificar o método que altera o número de pisos de forma a que o total de pisos especificado seja inferior a um dado valor.
- é possível ter muitos níveis de herança numa estrutura em árvore de heranças ou hierarquia de classes

Herança, porquê utilizar

- reutilizar código já existente ou usar bibliotecas de classes, o que permite aumentar a produtividade e desenvolver código com menos erros (uso de classes já testadas)
- vantagem POO: a reutilização de código; subclasses herdam os atributos e métodos da superclasse e basta apenas acrescentar métodos e variáveis para conseguir o comportamento especializado para a aplicação

Herança, porquê utilizar

- o JAVA faz uso intensivo de classes abstractas
  - superclasses que proporcionam comportamentos genéricos; todas as classes em JAVA são subclasses de uma superclasse
  - uma classe abstracta deixa muita da definição de classe não implementada de modo a que as subclasses possam definir comportamentos próprios
- O toolkit de JAVA proporciona vários grupos de classes JAVA (*packages*)
  - os pacotes são grupos de classes que realizam operações semelhantes. O AWT (*Abstract Windowing Toolkit*) é package que oferece um grupo de classes que podem ser usadas para visualizar e controlar janelas, botões, caixas de listagem, menus, entre outros.

A linguagem JAVA

- é orientada a objectos
  - semelhante, em sintaxe, ao C++
  - desenvolvimento de aplicações “stand alone”
  - desenvolvimento de aplicações para a Internet e multiplataforma
  - as aplicações para a Internet (Applets) são corridos pelos navegadores Web

A linguagem JAVA

- limitações
  - os applets JAVA são carregados de um host e corridos localmente
  - acesso a ficheiros locais é restringido em função da organização do sistema de ficheiros
  - acessos não autorizados de aplicações externas podem contaminar o disco local (vírus, …)

Applets e aplicações

- os **applets** diferem das aplicações por poderem ser executados páginas HTML, quando visualizados em navegadores web que suportam JAVA
  - possibilita aos programadores maior flexibilidade no conteúdo das páginas web. *Os applets podem correr em qualquer computador*
- as **aplicações** só correm no computador específico (ou família de computadores) para que foram compilados; não correm com o navegador
  - o JAVA é semelhante para desenvolver aplicações ou applets, mas existem diferenças criadas pelo ambiente no qual correm applets e aplicações

Limitações dos applets

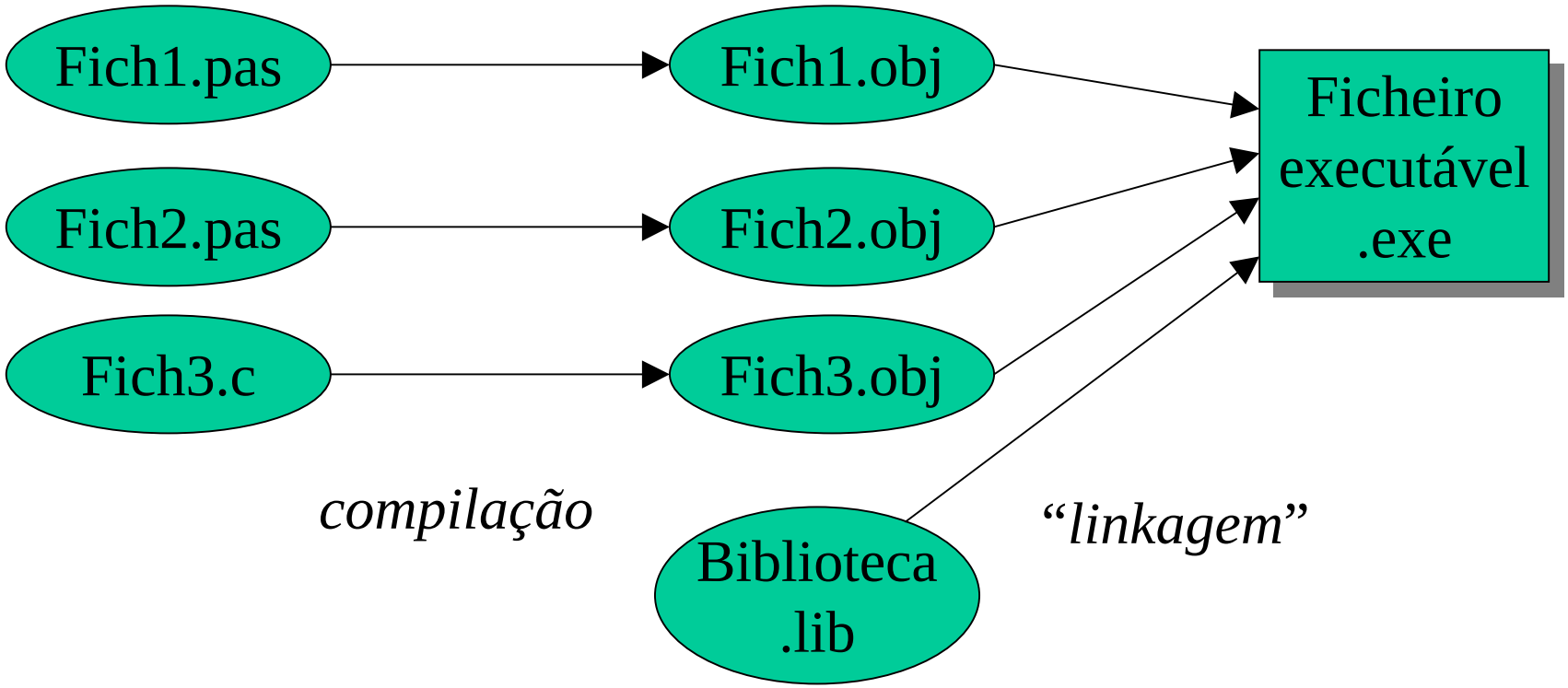
- podem não ser capazes de ler ou escrever ficheiros no computador local
  - além das questões de segurança, também existem questões de compatibilidade entre diferentes sistemas de ficheiros
- não podem carregar ou correr programas locais ou efectuar referências a DLLs
- não podem comunicacar com o servidor

Benefícios dos applets

- um bom interface de utilizador
- acesso à manipulação de acontecimentos
- acesso a um ambiente em rede
- capacidades gráficas melhoradas

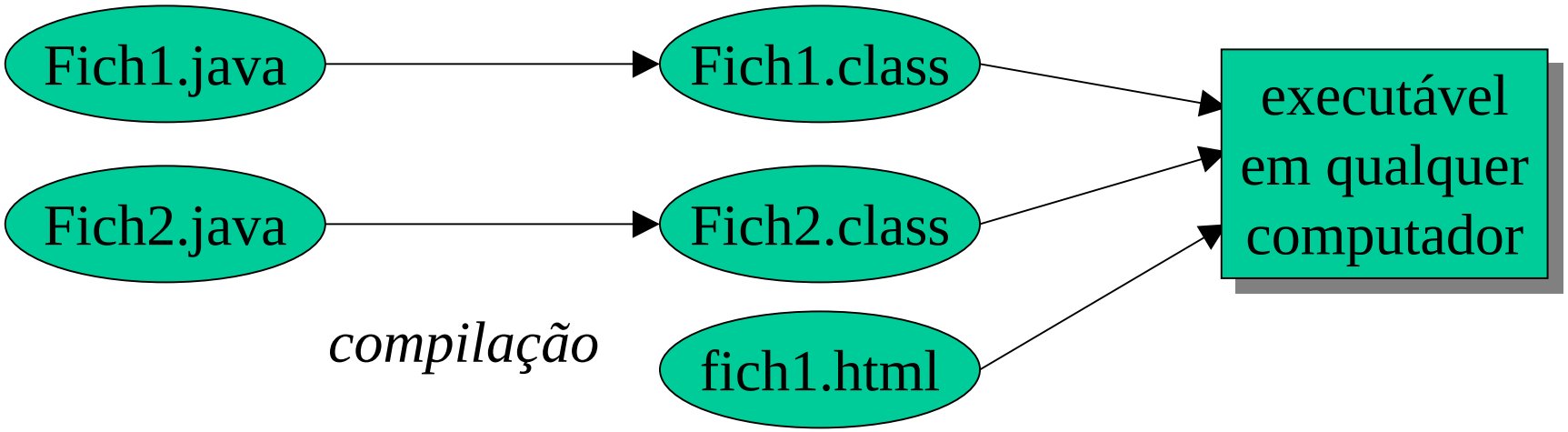
Diferenças dos applets

- A estrutura das aplicações tradicionais:



Diferenças dos applets

- A estrutura dos applets



Um applet simples

- estrutura de um applet
  - existem 5 métodos principais no JAVA (se não for incluida uma versão própria destes métodos, será usada a por defeito)
  - os métodos são:
    - init
    - start
    - stop
    - destroy
    - paint

...não existe um método main!

Inicializar um applet

- o primeiro método a ser executado quando se corre um applet
  - usado para inicializar valores, carregar imagens para posterior uso,...
  - o JAVA possui um método init que pode ser substituído por um a desenvolver

```
public void init( ) {
…
}
```

Iniciar um applet

- o método a ser executado a seguir ao init
  - pode ser chamado as vezes que se pretender, durante a vida do programa
  - quando se troca de página o applet pára de correr e quando se retorna à página, o método start é novamente executado

```
public void start( ) {
…
}
```

Parar um applet

- o método stop é o complemento do método start
  - pára de correr um applet
  - sempre que se sair da página, o applet continua a correr, no entanto pode-se parar a sua execução até se retornar à página

```
public void stop( ) {
…
}
```

Destruir um applet

- destruir o applet significa parar todos threads criados e libertar os recursos de computador que lhe foram alocados
  - normalmente, estas operações são realizadas pelo método por defeito e não existe necessidade de o modificar

```
public void destroy( ) {
…
}
```

Visualizar texto e gráficos

- Usado quando se visualiza texto ou gráficos
  - também é chamado quando se redesenha o ecran, como no caso de ser necessário redesenhar a janela do navegador por ter sido sobreposta outra janela de uma qualquer aplicação
  - método muitas vezes chamado durante a vida do programa de um applet típico (deve possuir bom desempenho)

```
public void paint(Graphics g ) {
…
}
```

O método paint

- possui um argumento; uma instância da class Graphics
  - o objecto é criado automaticamente
  - é necessário assegurar que a classe Graphics é importada para o applet, colocando no inicio do applet

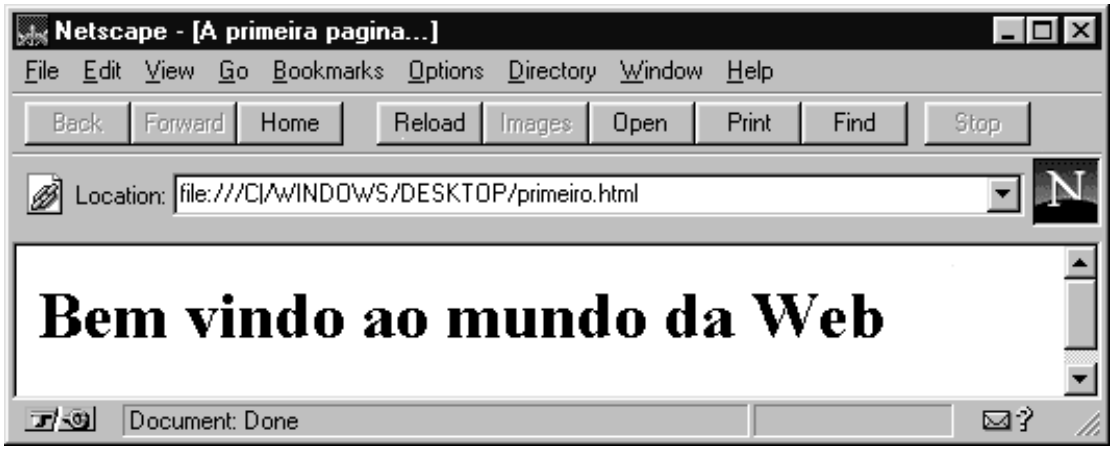
    import java.awt.Graphics
  - caso se queiram incluir subclasses (Color, por exemplo), utiliza-se

    import java.awt.Graphics.*

HTML

- As páginas web são escritas em HTML. Os items da linguagens, designados por tags, encontram-se entre "<" e ">":

```
<html>
<head>
<title>
A primeira pagina…
</title>
</head>
<body>
<h1> Bem vindo ao mundo da Web </h1>
</body>
</html>
```



Tags HTML

- Cada um dos tags é interpretado pelo navegador:
  - <HTML>, indica o iníco de um documento HTML
  - </HTML>, indica o fim de um documento HTML
  - <! …>, contém um comentário não visualizado pelo navegador
  - <HEAD> e </HEAD> são o iníco e fim da parte de metainformação do documento e contém informação sobre este, como é o caso do seu título
  - <TITLE> </TITLE>, visualiza o texto entre estes dois tags no topo da janela do navegador
  - <BODY> e </BODY> indica o início e o fim do documento a ser visualizado pelo navegador

O tag Applet

- os applets são incluidos nas páginas web recorrendo ao tag <APPLET>
  - quando o navegador carrega uma página que possui o tag <APPLET>, o applet é também carregado e executado
  - o tag <APPLET> possui diversas opções que controlam o seu tamanho e posição na página web, o alinhamento com o resto do texto e gráficos e os parâmetros a serem fornecidos ao applet

O tag Applet

- o tag <APPLET>
  - tem de estar entre o <BODY> e </BODY>

<BODY>

>APPLET CODE = “olamundo.class” WITH = 150 HEIGHT = 25>

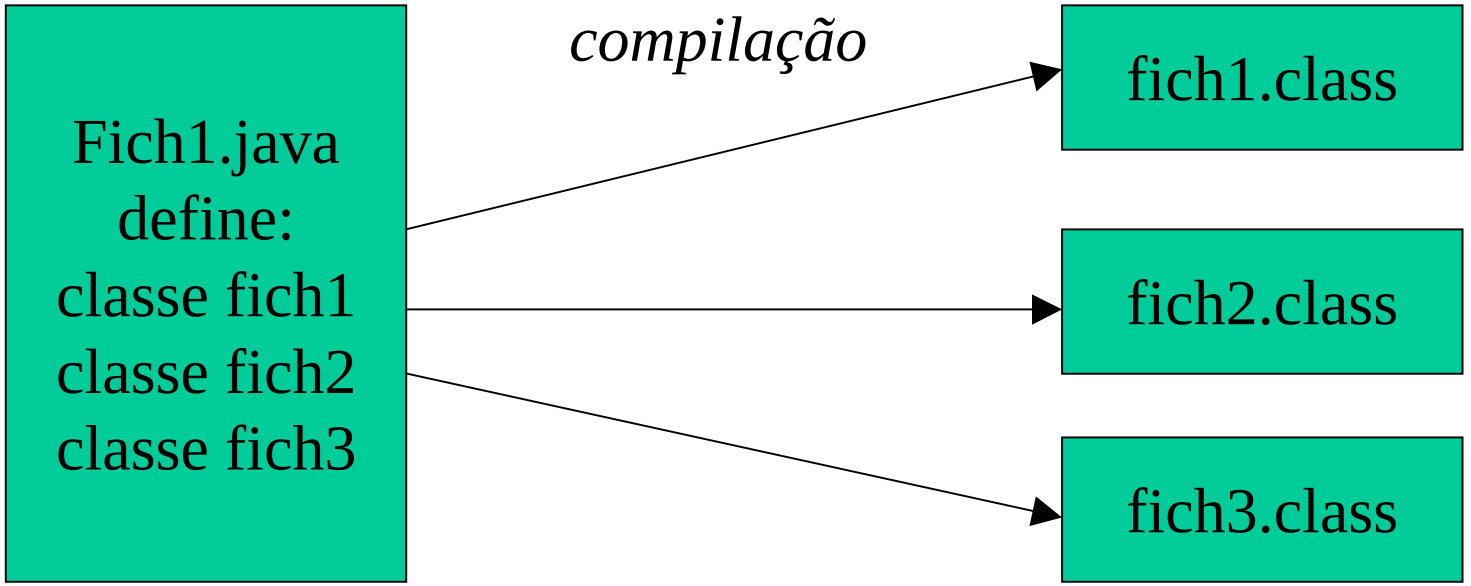
</APPLET>

</BODY>

  - o código fonte para o applet é o ficheiro olamundo.java; quando compilado cria o ficheiro de classe olamundo.class
  - os tags WITH e HEIGHT definem a área utilizada pelo applet, quando é executado

O tag Applet

- Se se definir mais de uma classe no código fonte, é produzido um ficheiro com a extensão .class para cada classe definida



O tag Applet

- o nome do ficheiro HTML deve ser o mesmo do ficheiro classe que inicia o applet
  - *olamundo.java*, ficheiro código fonte
  - *olamundo.class*, ficheiro classe produzido pela compilação do ficheiro olamundo.java
  - *olamundo.html*, ficheiro html, com o tag <APPLET>

Criação de um applet

- applet para colocar texto no navegador…

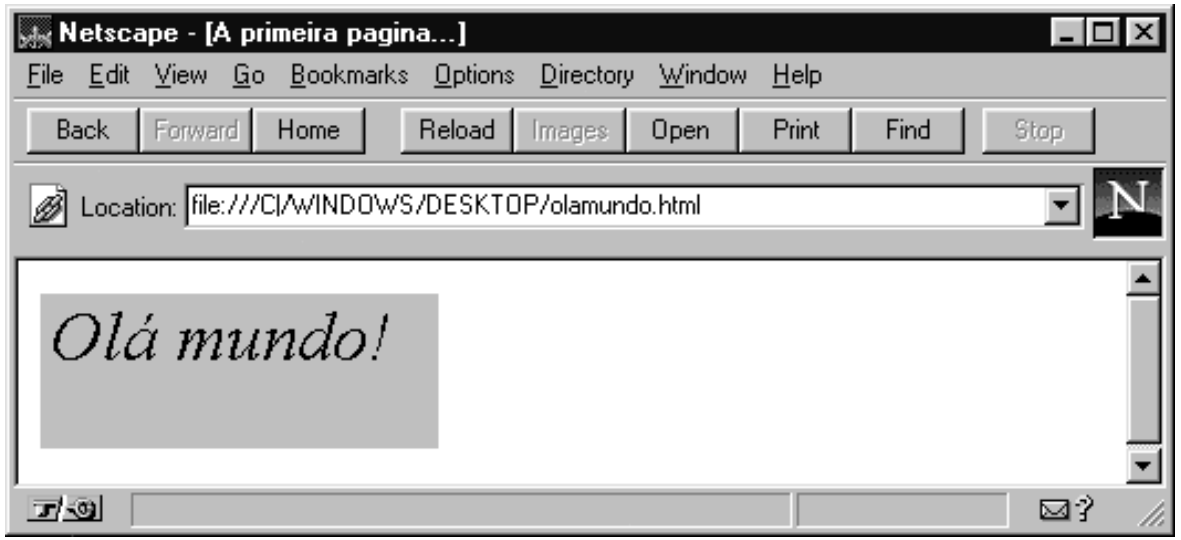
```
import java.awt.*;
public class olamundo extends java.applet.Applet {
   public void paint(Graphics g){
     Font escolha= new Font("TimesRoman",
  Font.ITALIC, 32);
     g.setFont(escolha);
     g.drawString("Olá mundo!",5,30);
   }
}
```

Processo de produção do applet

- O código fonte, *olamundo.java*, é compilado da seguinte forma:
  - **javac** *olamundo.java*
  - javac: ferramenta do jdk - *java development kit*, java compiler
- é produzido o código objecto: *olamundo.class*
  - é este ficheiro que é referido na página HTML, no tag <APPLET> e que é executado
- o ficheiro *olamundo.html* é criado com o tag descrito e pode ser visualizado num navegador, juntamente com o resultado da execução do applet

O ficheiro olamundo.html

```
<html>
<head>
<title>
A primeira pagina…
</title>
</head>
<body>
<applet code ="olamundo.class" width=180 height=70>
</applet>
</body>
</html>
```



Controlar a posição do applet na página web

<applet code ="olamundo.class" width=180 height=70>

- além das opções *width* (largura) e *height* (altura) também se pode controlar o *align* (alinhamento do applet: *left*, à esquerda e *right*, à direita)
  - o atributo *align*, do HTML, possui ainda mais 7 valores:
    - *middle*, centra o applet com o meio da linha de texto
    - *absmiddle*, o applet é alinhado com o meio do maior elemento da linha
    - *top*, alinhado o topo do applet com o topo da linha
    - *texttop*, alinhado o topo do applet com o topo do maior elemento da linha
    - *bottom*, o fundo do applet com o fundo do texto
    - *absbottom*, o fundo do applet com o menor elemento da linha
    - *baseline*, o mesmo que o *bottom*

Compilar aplicações

- uma aplicação desenvolvida em JAVA é compilada da mesma forma que um applet
  - **javac** aplicacao.java
- para correr a aplicação é utilizado o interpretador JAVA
  - **java** aplicacao

Argumentos para a linha de comando de uma aplicação

- Algumas aplicações necessitam de que lhes seja passada informação, quando são invocadas
  - argumentos da linha de comando
  - numa aplicação que ordene um conjunto de número, a aplicação é mais útil se se lhe der um ficheiro de entrada e o nome de um ficheiro para o resultado, como argumentos
- em JAVA existe apenas um parâmetro passado para a aplicação
    - args: um vector de strings contendo os argumentos
    - o número de argumentos pode ser verificado usando o método length( ) do vector

Argumentos para a linha de comando de um applet

- nas aplicações o método *main* é usado para a passagem de argumentos (nos applets não pode…)
- os parâmetros são passados nos applets via html , com o atributo *param*

```
<APPLET CODE=“palavras.class”>
<PARAM NAME=texto VALUE=“funciona?!”>
</APPLET>
```

Argumentos para a linha de comando de um applet

- também é necessário modificar o método *init* para receber o parâmetro com o método *getParameter*

```
import java.awt.Graphics;
public class palavras extends java.applet.Applet {
   String txtj;
   public void init ( ) {
     String txtj = getParameter("texto");
   }
   public void paint(Graphics g){
     g.drawString("Texto: " + txtj, 5, 30);
   }
}
```

Argumentos para a linha de comando de um applet

- é possível ter mais de um argumento de passagem entre html e o applet
- os parâmetros passados pelo html são strings

```
<APPLET CODE=“palavras2.class”>
<PARAM NAME=texto VALUE=“funciona?!”>
<PARAM NAME=numero VALUE=“3”>
</APPLET>
```

Argumentos para a linha de comando de um applet

```
/*  programa que recebe como parâmetros um texto e o número de vezes que
    o vai repetir, em linhas sucessivas*/
import java.awt.Graphics;
public class palavras2 extends java.applet.Applet {
   String txtj;
   int num;
   public void init ( ) {
     String txtj = getParameter("texto");
     String n=getParameter("numero");
     num=Integer.parserInt(n);
   }
   public void paint(Graphics g){
     for (c=0; c<num; c++) {
          g.drawString("Texto: " + txtj, 5, 20 + c*20);
     }
   }
}
```

JAVA

- O JAVA é sensível à utilização de maiúsculas
- convenção em JAVA
  - os nomes das classes possuem letras maiúsculas
  - métodos e objectos em minúsculas
- ignorado pelo compilador
  - espaços
  - tabs
  - comentários entre /* e */
  - comentários de linha //
- as linhas de programação em JAVA terminam com ;

Nomes em JAVA

- é necessário dar nomes a variáveis, classes e métodos
  - um nome tem de começar por uma letra ou por "_" ou "$"
  - os restantes caracteres podem ser qualquer caracter que seja visível
  - a excepção aos nomes são as palavras reservadas do JAVA

Elementos chave em JAVA

- As aplicações e os applets contém os seguintes elementos:
  - definições de **classes**
  - **objectos** (instâncias de uma classe definida)
  - **variáveis** de dados
  - **métodos**, que definem as operações a realizar com os dados

Classes

- Em JAVA tudo é classes ou descreve o que uma classe faz
  - não possui funções e procedimentos, mas sim blocos de classes
- as classes iniciam-se com a palavra reservada *class*, seguida pelo nome da classe e o seu conteúdo entre chavetas

```
class NovaClasse {
…
}
```

Subclasses

- no caso de uma subclasse da classe *NovaClasse*, é necessário utilizar a directiva *extends*

```
class OutraClasse extends NovaClasse {
…
}
```

- o conjunto de operações que podem operar uma classe designam-se por métodos
  - os métodos são definidos dentro da classe

Criação de objectos

- uma classe pode ser comparada a um formulário que descreve as caraterísticas de um conjunto de objectos
- após a criação de uma classe é possível definir objectos que são instâncias dessa classe; para criar um novo objecto:
  - o nome da classe de que se quer criar uma instância
  - o nome do objecto que se pretende criar
  - a palavra chave new
  - o nome da classe, seguida de parentesis, com os valores iniciais dentro deles

Criação de objectos, exemplos

- criar um novo objecto *minhaCasa* da classe *Construcao*

  Construcao minhaCasa = *new* Construcao( );

- criar um novo objecto valor

  Aleatorio valor = *new* Aleatorio( );

  ou

  Aleatorio valor;

  valor = new Aleatorio( );

- a palavra chave *new* cria a instância, alocando a memória necessária para o objecto, inicializando as variáveis e chamando o método *constructor* para o objecto

Variáveis em JAVA

- antes de se poder utilizar uma variável é necessário definir o seu tipo
  - também se pode atribuir um valor no momento da definição
- as variáveis locais são definidas dentro dos métodos
- o JAVA não possui variáveis globais disponíveis para todos os métodos
  - pode-se definir variáveis de instância e variáveis objecto (permite a partilha dentro das classes)

Tipos de dados em JAVA

- criar três variáveis de tipo inteiro

  int primeiro, segundo, terceiro;
- criar duas variáveis de tipo inteiro, uma delas inicializada

  int velho, novo=50;
- criar uma variável de tipo booleano (alocados em 8 bits)

  boolean simNao;
- o tipo caracter toma é representado em 16 bits contra os tradicionais 8 bits
  - recurso à representação de caracteres por um número de caracter Unicode

Tipos de dados

tipo	**descrição**	**número de bits**
byte	inteiro	8
short	inteiro	16
int	inteiro	32
long	inteiro	64
float	vírgula flutuante	32
double	vírgula flutuante	64
char	caracter	16
logical	booleano	8

- o tipo *String* não está incluído na lista de tipo de dados primitivos do JAVA
  - instância da classe *String* (logo, um objecto)
  - pode ser usado como um tipo de dados normal

Vectores

- os vectores são listas de itens
  - cada item é do mesmo tipo (um dos tipos primitivos)
  - em JAVA, os vectores são objectos (diferente das linguagens de terceira geração)
- declarar um vector

```
int valores[  ];
String cadeiaCaracteres[  ];
```

  - notação alternativa:

```
int [  ] valores;
String [  ] cadeiaCaracteres;
```

Vectores

- criar um objecto vector; duas formas:
  - usando *new*

    int [ ] valores = new int[5];
  - cria uma lista de 5 inteiros; o 1º referido por valores[0] e o último por valores[4]
- os vectores são inicializados, 0 para números, falso para booleanos, nulo para strings e \0 para caracteres
  - alternativamente pode-se especificar os seus valores iniciais

    int [ ] valores = {12, 9, 15, 18, 7}

    String [ ] lingProg = {“Java”, “C++”, “Pascal”};

Vectores

- atribuição de valores

  lingProg[2]=“Delphi”;

  – o JAVA verifica se o indice fornecido está dentro da gama de indices defina, pelo que caso tal não aconteça, ocorre um erro de excepção

- podem-se criar vectores multidimencionais

  – em rigor, tratam-se de vectores de vectores…

    int tabela[ ][ ] = new int[10][10];

  – o atribuição de valores é realizada de modo semelhante ao dos vectores

    tabela[3][4] = 26;

Métodos

- descrevem um conjunto de operações que podem ser realizadas com os dados, dentro de uma classe
- para definir um método, especifica-se:
  - modificadores
  - o tipo de objecto retornado pelo método
  - o nome do método
  - os parâmetros passados ao método
  - o corpo principal do método

Método, exemplo

- Exemplo de um pequeno programa com uma classe e um método:

```
class Programa {
    public static void main (String args[]) {
        System.out.println(“Saída de dados do programa”);
// coloca a mensagem no ecran
```

- o método *System.out.println* coloca uma string no ecran
- todas as aplicações JAVA (não os applets) tem que ter um método *main*

Métodos

- os modificadores podem ser ou públicos ou privados
  - um método público (*public*) pode ser chamado de qualquer lugar, na aplicação
  - um método privado (*private*) pode ser usado pela classe que o definiu
  - um método protegido (*protected*) pode ser usado em qualquer lugar, de onde foi definido
- se não se especificar se oo método é publico ou privado, é colocado o valor por defeito de *protected*

Métodos

- os métodos podem ser ou não estáticos (*static*):
  - os métodos **estáticos** possuem uma instância por classe e por isso todos os objectos que pertencem a essa classe partilham a mesma cópia do método
  - os métodos **não estáticos** possuem uma instância por objecto. Para especificar não estático, omite-se a palavra chave *static*
- o valor por defeito é não estático

Métodos de chamada

```
class ConvGraus {
    public static void main (String args []) {
                float centi, fahr = 67.5;
                centi = fahrParaCenti(fahr);
                System.out.println(fahr + "f = " + centi + " c");
                }
    }
class temperatura {
    public static float fahrParaCenti(float valor) {
    return (valor - 32)*5/9;
    }
}
```

- 2 classes e 1 método, que converte escalas de temperatura
- o valor retornado é do tipo vírgula flutoante (o tipo por defeito é o inteiro). Para o retorno do valor é utilizado o *return* (se o *return* é *void*, então nenhum valor é retornado)

Criar uma classe

- para criar uma classe e instânciar esta...
  - primeiro, definir uma classe

```
class Construcao {
}
```

- a seguir, definir as variáveis da classe:

```
class Construcao {
    int pisos;      //número de pisos da construção
    float area;     //área coberta de construção
    String uso;   //trabalho, armazém, lazer, residência
}
```

Criar uma classe

- terceiro; criar os métodos para a classe, por exemplo os dois métodos seguintes visualizam e modificam o uso da classe construção

```
void visualizaUso () {
        System.out.println(“A construção é usada para “ + funcao);
}

void modificaUso() {
      uso=novoUso);
}
```

Utilizar uma classe

- a declaração das variáveis e métodos definem a classe
- para criar uma instância de uma classe é necessário, no método *main* (*b* instância da classe *Contrucao*)

```
Contrucao b = new Contrucao( );
```

- para criar uma aplicação executável adiciona-se um método *main* depois da definição da classe

```
public static void main(String args[ ] ) {
    construcao b = new Construcao( );
    b.pisos=9;
    b.area=6500;
    b.uso=“residência”;
    b.visualizaUso( );
    b.modificaUso(“trabalho”);
    b.visualizaUso( );
}
```

Variáveis de instância

- As classes possuem variáveis associadas que fazem parte da sua definição
  - declaradas da mesma forma que as variáveis locais, mas definidas fora do método
  - designadas por variáveis de instância, são globais para cada instância da classe

```
class Armazem extends Construcao {
        int areaPiso;
        int numeroPortas;
        String uso;
}
```

Variáveis de classe

- definidas da mesma forma que as variáveis de instância, mas com a palavra chave static

  static peDireito

  – acessível globalmente na classe, em substituição de uma variável diferente por cada instância

- as variáveis de uma classe são colocadas após a definição da classe, junto à definição das variáveis de instância

Constantes

- variável cujo valor é fixo no início do programa e que não pode ser mudado

  final altura = 2.40;

  final String erroFich = “Ficheiro não encontrado”;

- em JAVA, as variáveis locais não podem ser constantes, apenas as variáveis de instância e de classe
- para delcarar uma constante é utilizada a palavra chave *final* e o valor pretendido

Uso de variáveis de instância

- as variáveis de classe são globais à classe
- **as variáveis de instância** são locais à instância

```
public static main (String args []) {
Armazem deposito = new Armazem( );
deposito.uso = “Guardar monos”;
deposito.areaPiso = 2000
deposito.numeroPortas = 4
…
}
class Armazem extends Construcao {
        int areaPiso;
        int numeroPortas;
        String uso;
}
```

Uso de variáveis de classe

```
Class Fiat {
        static String marca = “Fiat”;
        int ccMotor;
        …
}
   ...
   Fiat carroUtilitario, carroDesportivo
   carroUtiliario.ccMotor = 1300
   carroDesportivo.ccMotor = 2000
   System.out.println(“a marca do automovel é “ + carroUtilitario.marca);
   …
```

- *carroUtilitario* e *carroDesportivo* são duas instâncias da classe *Fiat* e podem possuir diferentes valores nas variáveis de instância *ccMotor*
- tem que ter o mesmo valor para a variável *marca* porque se trata de uma variável de classe (existe um valor por classe em vez de um valor por instância)

A palavra chave *this*

- possibilita a referência ao objecto corrente dentro de um método, usando a notação do ponto
  - se o objecto corrente possui uma variável de instância designada por *valor* é possível referir esta por *this.valor*
  - se se pretender passar o objecto corrente como um parâmetro, *this* proporciona um meio fácil de o fazer

    Inicializar(this)
- a palavra chave só pode ser usada dentro de um método de instância e não num método de classe, uma vez que *this* se refere à instância corrente de uma classe

A aritmética e o JAVA

- o JAVA possui cinco operadores aritméticos:
  - + (adição); - (substracção);
  - / (divisão); * (multiplicação) e % (módulo)
- o mesmo operador é usado para a divisão de inteiros e reais
  - o resultado da divisão inteira não é arredondado 14/5 = 2 e não 3
- o operador módulo, %, calcula o resto de uma divisão inteira
  - o resultado de 14%5 é 4

Atribuição de variáveis numéricas

- a atribuição é realizada com o operador =
- com variáveis de tipos diferentes, existe uma conversão de tipo antes de serem operadas essas variáveis
  - *int* e *float*; as variáveis inteiras são convertidas para *float*
  - *float* e *double*; as variáveis do tipo *float* são convertidas para double
- as variáveis mais fracas (de menor gama de valores) são convertidas para as variáveis mais fortes, presentes como variáveis a operar

Atribuição de variáveis numéricas

```
double litro, galao;
float milha, kilometro;

litro=galao*0.22;
kilometro = (milha*5)/8;
```

- 0.22 é automaticamente uma quantidade de tipo *double*
- as constantes 5 e 8, inicialmente inteiros, são convertidas para *float* por causa da variável *milha*: kilometro = (milha * 5.0) / 8.0

- após os cálculos, o resultado é convertido para o tipo da variável do lado esquerdo da atribuição

Atribuição de variáveis numéricas

```
int centigrados, pontoCongelacao;
double fahrneit;

pontoCongelacao = 32;
centigrados = (fahrneit - pontoCongelacao) *5.0/9.0;
```

- o tipo de dados mais forte é o *double*, atribuído à variável *fahrneit*
- o cálculo é realizado com todos os valores convertidos para *double*
- o resultado é então convertido para o tipo *int*, da variável *centigrados*

Casting de variáveis

- quando se pretende alterar a conversão automática entre diferentes tipos que o JAVA possui por defeito utiliza-se o *casting* de variáveis
  - especifica-se o tipo de variável para o qual se pretende a conversão

```
int c, d;
float valor;
...
valor = (float) c / (float) d;
```

  - converte as variáveis *int* em variáveis do tipo *float*
- pode-se efectuar o *cast* de um objecto para outro que seja sua superclasse ou subclasse
- não é possível realizar o *cast* de um objecto para um dos tipos de dados primitivos

Atribuição de variáveis não numéricas

- para atribuir um caracter

```
letra = ‘L’;
caracterNovaLinha = ‘\n’;
```

  - existem vários caracteres de controlo especiais que o JAVA permite especificar

\n	nova linha	\\	caracter \
\t	tecla *tab*	\’	caracter ‘
\b	*backspace*	\”	caracter ”
\r	tecla *return*	\ddd	número octal
\f	*formfeed*	\xdd	número hexadecimal
\udddd	caracter *unicode*		

Atribuição de variáveis não numéricas

- para o tipo booleano, podem-se atribuir os valores *true* e *false*:

  boolean campeaoFCP = true;

- para o objecto String, a atribuição é realizada com o valor entre aspas:

  texto = “Esta é uma pequena frase”;

- no objecto String também se é possível utilizar os códigos dos caracteres especiais:

  texto = “1. Primeira parte \n\t1.1. Introdução\n”;

Operadores de atribuição

- além do operador =, o JAVA possui outros

  contador1, contador2, contador3 = 0:

  – é atribuído o valor zero a todas as variáveis
- existe uma forma abreviada de escrever as seguintes expressões

  ```
  contador1 = contador1 + 7;
  novoValor = novoValor / intervalo;
  ```
- é a seguinte:

  ```
  contador1 += 7;
  novoValor /= intervalo;
  ```

Operadores de atribuição

operação	significado
c += d	c = c + d
c -= d	c = c - d
c /= d	c = c / d
c *=d	c = c * d
c++	c = c + 1
c--	c = c - 1

Operadores de atribuição

- no caso de se incrementar (ou decrementar, neste caso, substituir o sinal + pelo sinal -) um valor é possível representar a operação de uma das seguintes quatro formas:

```
contador1 = contador1 + 1;
contador1 += 1;
contador1 ++;
++ contador1;
```

Atribuição de objectos

- O operador = é usado para atribuir as variáveis de um objecto a outro

```
public void class Emprego {
String ocupacao;
int vencimento;
…
empPaula.ocupacao = emp.Sara.ocupacao;
emp.vencimento = emp.Sara.ocupacao;
…
}
```

Atribuição de objectos

- não é possível ter

  empPaula = empSara

- para copiar todas as variáveis de instância de um objecto é utilizado o método copy:

  empPaula = copy(emp.Sara)

Operadores de comparação

operação	significado
==	equivalente
!=	diferente
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

Operadores de comparação

- diferença entre o operador de atribuição e o operador de equivalência:

  if (c = = d) e = f;

  – compara *c* e *d*

  – se *c* e *d* forem iguais, atribui o valor de *f* a *e*

- no entanto a expressão seguinte, não tem sentido:

  if (c = d) e = = f

  - o valor de *d* é atribuido a *c* e nunca será atribuido o valor *f* a *e* porque *e* = =*f* é um teste de igualdade

Comparação de objectos

- podem-se usar os operadores = = e *!* =
  - cuidado; estes operadores não testam se os valores atribuidos aos dois operandos são iguais!
  - com diferentes objectos *String* que possuem o mesmo texto o valor retornado não é *true*, quando se testa a igualdade, a não ser que ambos refiram o mesmo objecto

Sobreposição de métodos

- o que é?
  - na definição de um método é necessário declarar explicitamente a lista de parâmetros que lhe são passados
  - no JAVA, quando um método é chamado com uma lista de pârametros incompleta ou incorrecta, é obtido um erro
  - com parâmetros opcionais, define-se um método com o mesmo nome mas com uma lista diferente de parâmetros
  - quando é chamado um método, o JAVA *examina os parâmetros fornecidos e chama o método apropriado*

Sobreposição de métodos

```
Class MostraErro {
    MostraErro relataErro(String menasagem) {
        System.out.println(mensagem);
    }
    MostraErro relataErro( ) {
        System.out.println(“Erro desconhecido.”);
    }
}
```

– pode-se chamar o método *relataErro* com um parâmetro ou sem prâmetro

Construtores

- métodos especiais que controlam como um objecto é inicalizado
- não podem ser chamados directamente, mas são lançados pelo JAVA sempre que um objecto é criado (sempre a directiva new é utilizada)
- podem ser usados por feitos ou definidos pelo utilizador
- um método construtor para a classe possui o mesmo nome desta mas não tem tipo de retorno

Construtores, exemplo

- com os construtores por defeito

```
class Pais {
    String nome;
    String capital;
    int populacao;
public static void main(String args[ ] ) {
    Pais portugal;
    portugal = new Pais( );
    portugal.nome = “Portugal”;
    portugal.capital = “Lisboa”;
    portugal.populacao = 8800000
    System.out.println(“A capital de” + portugal.nome);
    System.out.println(“ é “ + portugal.capital);
    System.out.println(“A população é” + portugal.populacao);
   }
}
```

Construtores, exemplo

- com o construtor *Pais*

```
class Pais {
        String nome;
        String capital;
        int populacao;
Pais(String oNome, String aCapital, int aPopulacao) {
        nome = oNome;
        capital = aCapital;
        populacao = aPopulacao;
}
public static void main(String args[ ] ) {
                Pais portugal;
                portugal = new Pais( Portugal, Lisboa, 8800000);
                System.out.println(“A capital de” + portugal.nome);
                System.out.println(“ é “ + portugal.capital);
                System.out.println(“A população é” + portugal.populacao);
        }
}
```

O método *finalize*

- chamado quando um objecto é destruido ou quando um programa termina abruptamente
- só é criado pelo utilizador quando este pretende optimizar a gestão de memória, assegurando que todos os recursos de memória atribuidos ao objecto foram libertados
- para criar um método *finalize* próprio

```
void finalize(  )  {
    …
}
```

Tratamento de condições

- tomar decisões…
  - *if…else*

```
if (memoria < 16)
        System.out.println(“Está tudo um bocado lento?”);
```

  - com a clausula *else*

```
if (memoria < 16)
        System.out.println(“comprar mais memória”);
else
        System.out.println(“memória suficiente”);
```

Tratamento de condições

– vários *if…else* encadeados

```
if (memoria < 8)
        System.out.println(“necessário comprar mais memória”);
else if (memoria <=16)
            System.out.println(“memória suficiente”);
        else
            System.out.println(“memória adequada para trabalho”);
```

- várias condições a testar

```
if ((dia = = domingo) && (canal = = 1))
        System.out.println(“podemos ter futebol na televisão...”);
```

Blocos de código

- permite o agrupamento de linhas de programação em blocos:

```
if (dia = = “sabado”)
fazerCompras(  );

if (dia = = “sexta) {
    calcularVencimentos( );
    deduzirImpostos( );
    imprimirCheques( );
    }
```

O operador condicional

- o seguinte código

```
if (memoria <= 16)
mensagem = “memória pequena);
else
mensagem = “memória adequada”;
System.out.println(mensagem)
```

- é equivalente a

```
String mensagem = memoria <=16 ? “memória pequena” :
   “memória adequada”
System.out.println(mensagem)
```

- sintaxe geral:
  - test ? Resultado verdadeiro : resultado falso

A estrutura *switch*

- forma de especificar múltiplas opções:

```
if (memoria = = 8)
  System.out.println(“necessário mais memória”);
else if (memoria = = 16 )
  System.out.println(“possui a memória suficiente”);
else if (memoria = = 32)
  System.out.println(“possui a memória aconselhável”);
else if (memoria = = 64)
  System.out.println(“tem memória para trabalhar bem”);
```

  - com *switch*, é possível reescrever o código...

A estrutura *switch*

```
Switch (memoria) {
  case 8:
     System.out.println(“necessário mais memória”);
    break;
  case 16:
     System.out.println(“possui a memória suficiente”);
     break;
  case 32;
     System.out.println(“possui a memória aconselhável”);
     break;
  case 64:
     System.out.println(“tem memória para trabalhar bem”);
}
```

A estrutura *switch*

- a clausula *break* assegura que na estrutura *switch* cada bloco asociado a um *case* é realizado, sem os seguintes serem executados
- a clausula *default* pode ser usada no final de todas as opções *case*; neste caso, se nenhuma outra linha de código for executada, o bloco *default* é executado
- existem algumas limitações para o
  - só se usar para condições de igualdade
  - só se podem usar os tipos *byte*, *char*, *int* e *short* para os testes