Aula 1 – Apresentação e Introdução

Algoritmos e Estruturas de Dados

Funcionamento

Da Unidade Curricular

Corpo Docente e Aulas

- Teóricas:
  - Docente: João Dias
  - Sala: 2.65
  - Email: jmdias@ualg.pt

- Práticas:
  - Docentes:

    João Dias (PL3)

    Nestor Cataño (PL1, PL2, PL4)

  - Aulas de resolução de exercícios práticos
  - Aulas de apoio aos projetos de programação
  - Aulas de validação dos projetos de programação

Avaliação

- Componente de avaliação contínua – 40%
  - **Participação nas aulas práticas**

    10%

    As aulas de laboratório começam já esta semana (hoje PL3)
  - Projectos de programação (4) ao longo do semestre

    Projeto 1 – 15% (Introdução ao java)

    Projeto 2 – 25%

    Projeto 3 – 25%

    Projeto 4 – 25%
  - Nota mínima na nota final: 8,0

- Exame – 60% (Exame escrito, componente teórico-prática)
  - Nota mínima: 8,0

UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Aproveitamento nota prática

- Alunos com nota prática positiva 2024/2025
  - Podem **aproveitar a nota para este ano**
  - **Não é obrigatória** a participação nos turnos PL
  - **Inscrevam-se PL9**
  - **Enviem-me email para jmdias@ualg.pt a pedir o aproveitamento da nota**

- Mas podem fazer projetos?
  - Sim, desde que não nos deem trabalho a avaliar.
  - Podem submeter no Mooshak à vontade

Aula de validação

- Na aula de laboratório a seguir à entrega do projeto, todos os alunos deverão validar o código submetido.
  - Breve análise e discussão sobre o código entregue, onde podem ser colocadas questões para aferir o conhecimento do aluno sobre o projeto e matéria teórica relacionada
  - 10 – 15 min
  - Participação é obrigatória
  - A nota do projeto poderá ser afetada pela prestação do aluno na validação

Deteção de cópias

- Do código de Honra:
  - “Os alunos são encorajados a discutir os problemas com outros alunos…”
  - “Mas os alunos não deverão copiar código de outros alunos, ou dar o seu próprio código a outros em qualquer circunstância…”

Código de Honra

- Subscrevam o código de Honra na tutoria

Código de Honra

Nesta disciplina, espera-se de cada aluno que subscreva os mais altos padrões de honestidade académica e integridade moral. De acordo com o Despacho n.º 2131/2020 - Código de Ética da Universidade do Algarve, os alunos comprometem-se a não incorrer em práticas fraudulentas, tais como:

a) Obter previamente enunciados de provas de avaliação, formulários, questionários ou outros elementos constantes das provas de avaliação com o intuito de daí resultar benefício para o próprio ou para terceiros;
b) Utilizar elementos não autorizados na prestação de provas em proveito próprio ou em benefício de outrem;
c) Copiar o trabalho de outrem ou permitir que outro copie o seu trabalho, receber ou dar ajuda a outro estudante em provas de avaliação;
d) Atuar como substituto ou fazer-se substituir por outro em provas de avaliação;
e) Praticar plágio, ou seja, utilizar ideias, afirmações, dados, imagens, parágrafos ou texto completo que não sejam da sua autoria, sem a adequada referenciação nos trabalhos, designadamente, de carácter literário, científico ou artístico;
f) Apresentar trabalho, realizado em conjunto com outro(s), sem que tal seja permitido;
g) Apresentar como novo e original, um trabalho já realizado e avaliado anteriormente;
h) Falsificar, omitir voluntariamente ou interpretar tendenciosamente dados e resultados na realização de trabalhos;
i) Falsificar assinatura e/ou informações em documentos oficiais;
j) Comercializar, no todo ou em parte, trabalho académico a favor de outrem, utilizado ou a utilizar em processo de avaliação.

Adicionalmente, a utilização de ferramentas de geração automática de código, como o CHAT-GPT, para gerar uma parte substancial de uma prova de avaliação corresponde a uma prática fraudulenta, de acordo com a alínea e).

Nesta disciplina, o método de avaliação inclui vários projetos e exame. Os alunos são encorajados a discutir os problemas com outros alunos e devem mencionar essa discussão quando submetem os resultados. Essa menção NÃO influenciará a nota. Mas os alunos não deverão copiar código de outros alunos, ou dar o seu próprio código a outros em qualquer circunstância. De facto, não devem sequer deitar listagens fora sem primeiro as destruir, nem deixar o código desenvolvido em computadores de uso partilhado.

Nesta disciplina, a desonestidade académica é considerada fraude, com todas as consequências legais que daí advêm. Qualquer fraude terá como consequência imediata a reprovação de todos os alunos envolvidos (incluindo os que possibilitaram a ocorrência). Qualquer suspeita de desonestidade académica será relatada aos órgãos superiores da escola para instauração de um processo disciplinar. Este poderá resultar em reprovação à disciplina, reprovação de ano, suspensão temporária ou definitiva da Faculdade de Ciências e Tecnologia ou mesmo da Universidade do Algarve.

Deteção de cópias

- Todos os projetos serão passados por um sistema de deteção de cópia
  - Não basta alterarem o nome de métodos e variáveis para enganar o sistema
- Casos potencialmente suspeitos serão analisados pelo corpo docente

Importante
No caso de cópia, quer o aluno que copiou, quer o aluno que deixou copiar terão a sua prova anulada

Bibliografia

- Bibliografia principal
  - Algorithms, 4th Editions (2011)

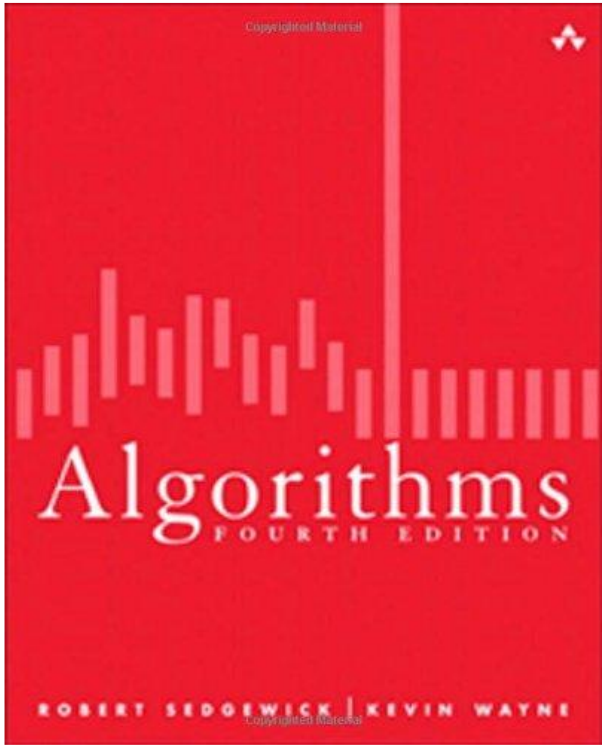
    Robert Sedgewick & Kevin Wayne

  - Book site:

    https://algs4.cs.princeton.edu/home/
  - From amazon:

    https://www.amazon.co.uk/Algorithms-Robert-Sedgewick/dp/032157351X/
  - Online pdf

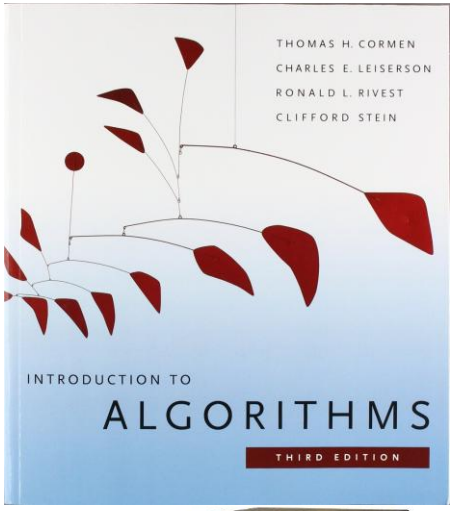
    https://mrce.in/ebooks/Algorithms%204th%20Ed.pdf



Bibliografia

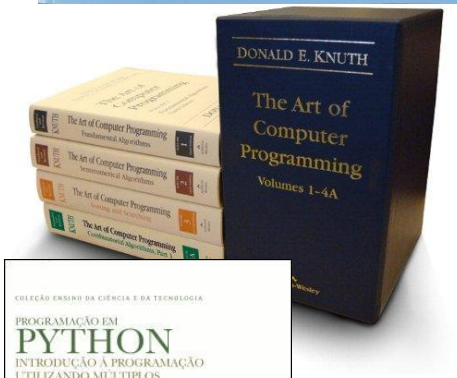
- Bibliografia secundária
  - Introduction to Algorithms, 3rd Editions

    Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein



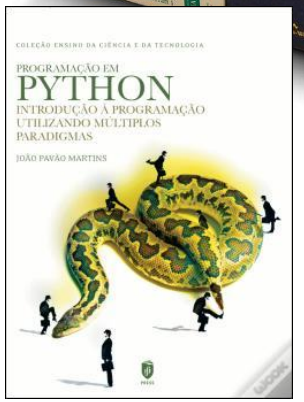
  - The Art of Computer Programming

    Donald Knutch



  - Programação em Python. Introdução à programação utilizando múltiplos paradigmas.

    João Pavão Martins



Tecnologia

- Linguagem de Programação:
  - https://www.oracle.com/java/technologies/downloads/
    JDK 17 ou mais recente

- Ambiente de Desenvolvimento:
  - Intellij IDEA Community Edition
    https://www.jetbrains.com/idea/
  - (Alternativamente) Eclipse
    https://www.eclipse.org/downloads/

- Sistema Operativo: Windows, MacOS, Linux
- Avaliador Automático: Mooshak

AED?

No que consiste esta cadeira?

Algoritmos e Estruturas de Dados

- Mas afinal, no que consiste esta cadeira?
- Algoritmos?
- Estruturas de Dados?

Algoritmos

- Algoritmo
  - ***Def:*** conjunto ou sequência finita de instruções/ações executáveis que visam obter uma solução para resolver um determinado problema

    The Editors of Encyclopaedia Britannica

Curiosidade:
o nome algoritmo deriva da tradução para latim do nome de um matemático persa do século IX: Muḥammad ibn Mūsā **al-Khwārizmī**.

Um dos livros publicados por ele foi *"Al-Khwarizmi concerning the Hindu Art of Reckoning"*, traduzido em latim para *"Algoritmi de numero Indorum"*.

Algoritmos

- Programa de computador é um caso particular de um algoritmo:
  - Diz ao computador os passos específicos e ordem pela qual devem ser executados para calcular uma resposta para um determinado problema
  - Portanto, vocês como alunos do 2.º ano, já fizeram e implementaram algoritmos

Desafio

- Aquilo que vão aprender nesta cadeira vai ser algo que vos vai tornar em algo mais do que programadores
  - Eng. Informáticos
- Vão aprender a analisar propriedades de um dado algoritmo
  - Complexidade Temporal
  - Complexidade Espacial
- Vão aprender e estudar famílias de algoritmos extremamente eficientes para determinados tipos de tarefas
- … e para determinadas estruturas de dados

Estruturas de Dados

- Estrutura de dados – forma de agrupar dados

- Vocês já usaram muito Tipos Estruturados em Laboratório de Programação

```
typedef struct S_Time {
    int hours;
    int minutes;
} Time;
```

Definição do Tipo Estruturado *Time* em *C*

Estruturas de Dados

- Até agora, vocês trabalharam principalmente com estruturas de dados onde
  - Dados guardados de forma linear em memória
  - Não tiram partido da relação que pode existir entre duas instâncias do mesmo tipo

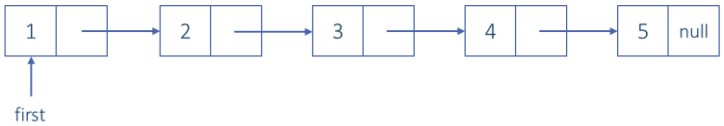
```
typedef struct S_Time {
    int hours;
    int minutes;
} Time;
```

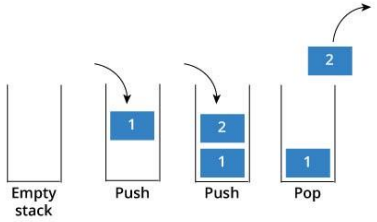
Estruturas de Dados

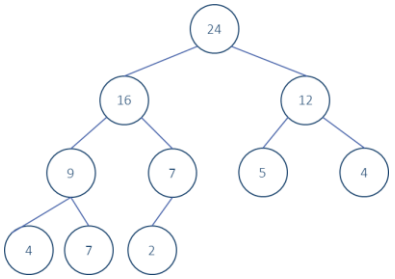
- Nesta cadeira iremos estudar Estruturas de Dados com as seguintes características
  - Especialmente desenhadas para guardar uma grande quantidade de objetos do mesmo tipo
  - Tiram partido de relações existentes entre os objetos para tornar muito eficientes certos tipos de operações sobre esses objetos

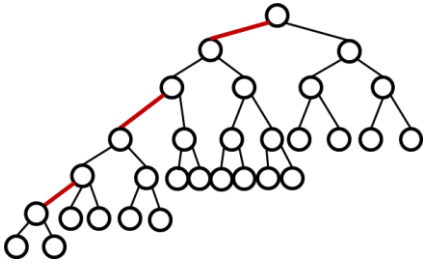
Estruturas de Dados

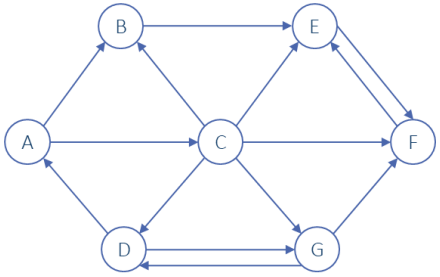
- Exemplos
  - Listas Ligadas
  - Filas, Pilhas
  - Amontoados (Heaps)
  - Árvores Binárias de Pesquisa
  - Tabelas de Dispersão
  - Grafos

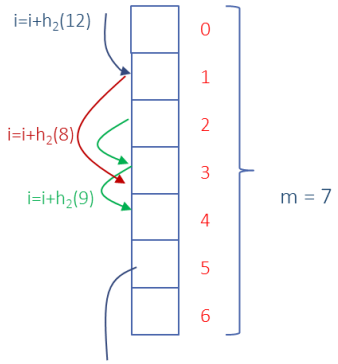












Introdução ao Java

Mas… antes de vos ensinarmos sobre Algoritmos e Estruturas de dados.

Vamos ter que fazer uma introdução à Linguagem Java

Características

da Linguagem Java

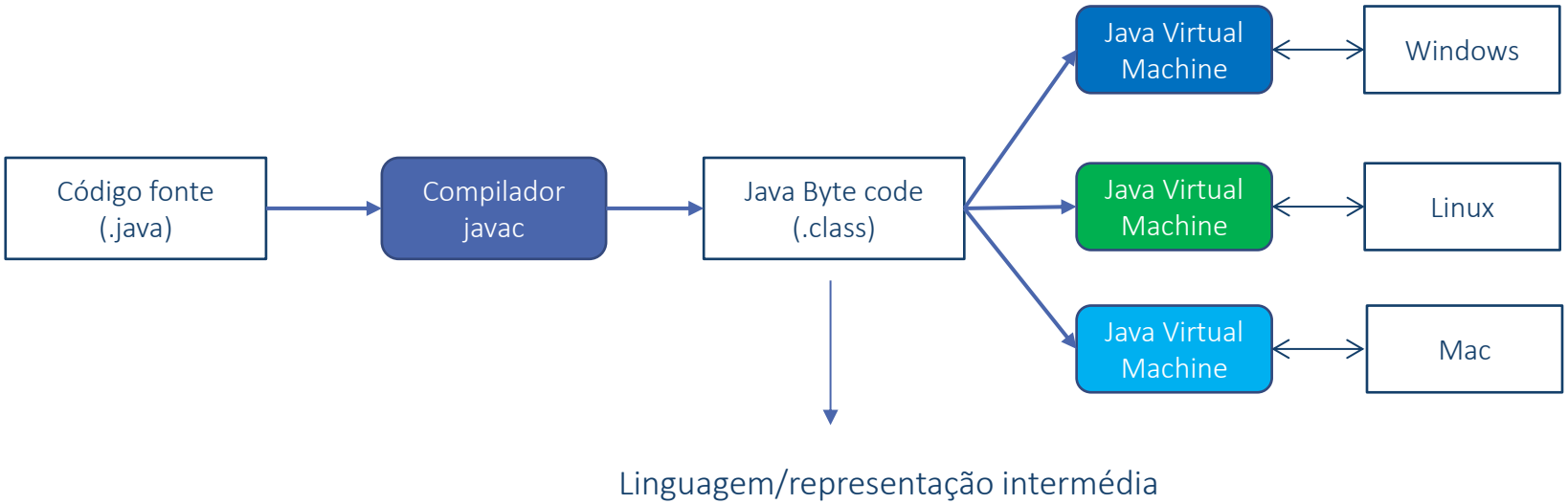
Linguagem de Programação Java

- Criada em 1995 por James Gosling na Sun Microsystems
  - Sun Microsystems adquirida mais tarde pela Oracle

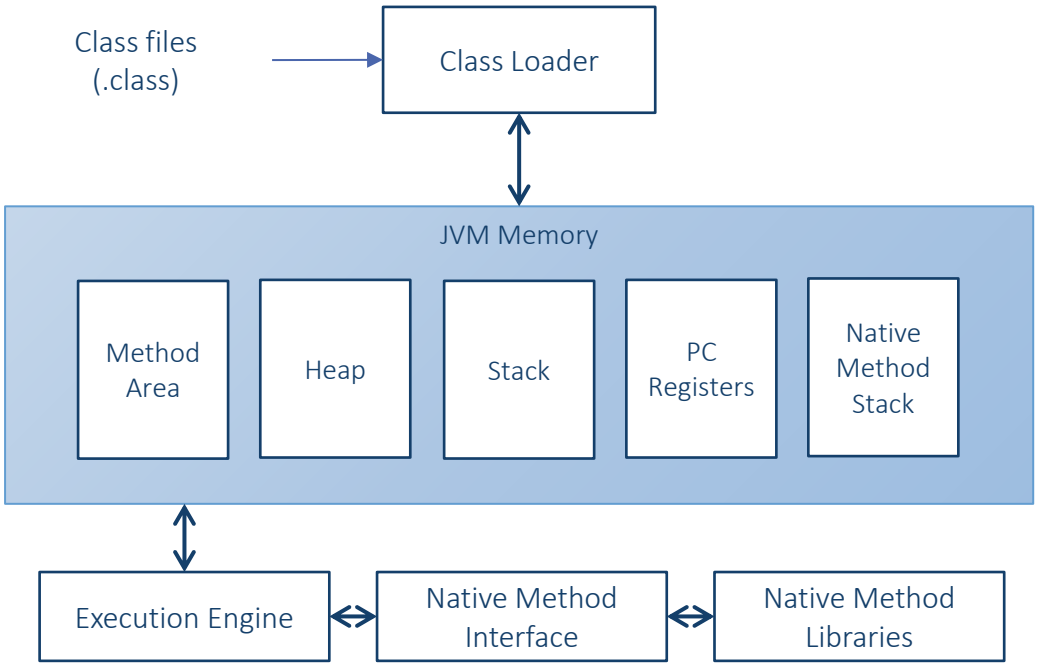
- Características principais
  - Portável e independente da plataforma de execução
    "Write once, run anywhere"
  - Simples, Orientada a objectos, e familiar
    Sintaxe parecida ao c/c++
  - Robusta e segura
    "Type safety"
    Gestão automática de memória
  - Linguagem Interpretada, com múltiplas "threads" e dinâmica
  - Executável com alta performance
    "Just In Time Compiler"

Portabilidade

- Compilador de java compila o código para uma linguagem intermédia

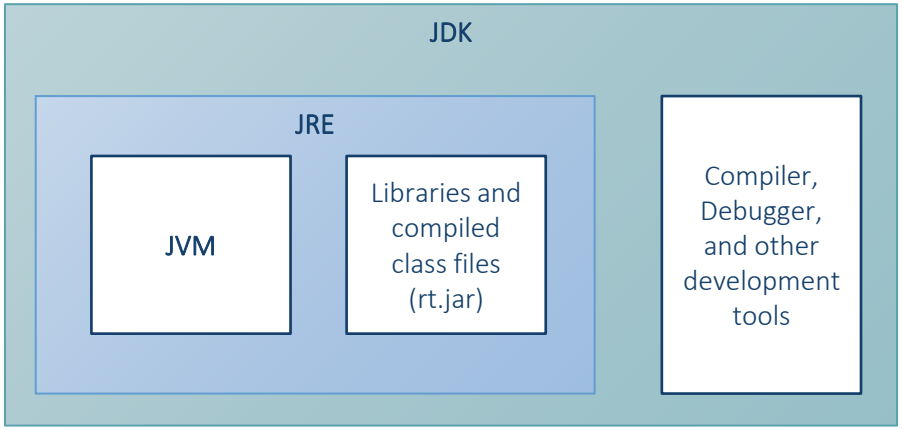


Java Virtual Machine



JDK/JRE

- JRE – Java Runtime Environment
  - Executa código binário java
- JDK – Java Development Kit
  - Ferramentas de desenvolvimento em java
- JAR – Java Archive. Ficheiro zip com código compilado java (java byte code)



Orientada a Objectos

- Java é uma linguagem que segue o paradigma Orientado a Objectos
  - Classe = definição de um TAI: dados + comportamento
  - Objecto = instância de uma classe

```
public class ContaBancaria {
	protected long NIB;
	protected double saldo;

	public ContaBancaria(long nib)
	{
		NIB = nib;
		saldo = 0;
	}
	public double levantamento(double valor)
	{
		if(saldo >= valor)
		{
			//levantamento efectuado com sucesso
			saldo -= valor;
		}
		return saldo;
	}
	public double deposito(double valor)
	{
		if(valor > 0)
		{
			saldo += valor;
		}
		return saldo;
	}
}
```

```
public class Main {

	public static void main(String[] args)
	{
		ContaBancaria conta = new ContaBancaria(2923948);
		conta.deposito(10.0);
	}
}
```

Robustez e Segurança

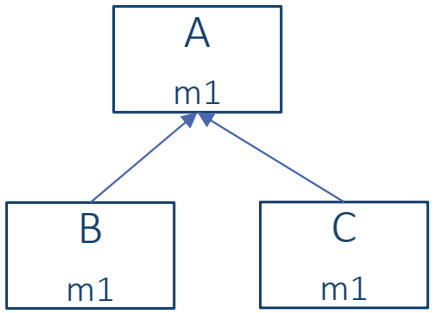
- Foram tomadas uma série de decisões que tornam mais limitado o que o programador pode fazer
  - Mas tornam a linguagem mais robusta, segura, e fácil de usar

- Ex:
  - Gestão automática de memória
  - Não existe aritmética de ponteiros
  - Deteção de acessos inválidos a vetores
  - Deteção de conversões de tipos (casts) inválidas

Linguagem Interpretada e dinâmica

- Interpretada (ou compilada *just in time*)
  - Código java bytecode é traduzido em *runtime* para instruções nativas na JVM
- Dinâmica
  - Java consegue executar funções cujo método exato só é determinado em execução

Ex: utilização de Polimorfismo




```
A a = new A();
B b = new B();
C c = new C();
A ref;

ref = a;
ref.m1(); // calling A's version of m1()

ref = b;
ref.m1(); // calling B's version of m1()

ref = c;
ref.m1(); // calling C's version of m1()
```

Performance em Java

- Java tem muitas propriedades interessantes

- Preço a pagar?

  Eficiência. C consegue ser mais eficiente.

- Melhorias significativas
  - JIT – Just In Time Compiler

    Vai compilando e optimizando o código à medida que este vai sendo executado.

Introdução

à linguagem Java

Instruções em Java

statement	examples	definition
declaration	`int i;` `double c;`	create a variable of a specified type, named with a given identifier
assignment	`a = b + 3;` `discriminant = b*b - 4.0*c;`	assign a data-type value to a variable
initializing declaration	`int i = 1;` `double c = 3.141592625;`	declaration that also assigns an initial value
implicit assignment	`i++;` `i += 1;`	`i = i + 1;`
conditional (`if`)	`if (x < 0) x = -x;`	execute a statement, depending on boolean expression
conditional (`if-else`)	`if (x > y) max = x;` `else max = y;`	execute one or the other statement, depending on boolean expression

Instruções em Java

statement	examples	definition

loop (while)	`int v = 0;`
	`while (v <= N)`
	`  v = 2*v;`
	`double t = c;`
	`while (Math.abs(t - c/t) > 1e-15*t)`
	`  t = (c/t + t) / 2.0;`
		execute statement
		until boolean expression is `false`

loop (for)	`for (int i = 1; i <= N; i++)`
	`  sum += 1.0/i;`
	`for (int i = 0; i <= N; i++)`
	`  StdOut.println(2*Math.PI*i/N);`
		compact version of `while` statement

call	`int key = StdIn.readInt();`	invoke other methods (see page 22)
return	`return false;`	return from a method (see page 24)

Tipos primitivos

Tipo	Descrição	Tamanho
byte	Número Inteiro	8 bit
short	Número Inteiro	16 bit
int	Número Inteiro	32 bit
long	Número Inteiro	64 bit
float	Número Real	32 bit
double	Número Real	64 bit
boolean	**valor true ou false**	**1 bit**
char	Caracter	16 bit

Tipos primitivos são sempre passados por valor (i.e copiados) quando usados na invocação de uma função

Vetores (arrays) em Java

- Criação de um vetor
  - Declaração
  - Criação
  - Inicialização

    Por omissão

    Tipos numéricos são inicializados a 0

    Tipo boolean a false

long form

```
double[] a;
a = new double[N];
for (int i = 0; i < N; i++)
   a[i] = 0.0;
```

declaration

creation

initialization

short form

```
double[] a = new double[N];
```

initializing declaration

```
int[] a = { 1, 1, 2, 3, 5, 8 };
```

Observação:
Ao contrário do C, onde podemos alocar vetores no *stack* ou no *heap*, em Java:

Os vetores são sempre alocados no heap.

Aliás, qualquer objeto computacional criado com a primitiva *new* é alocado sempre no *heap*!

Vetores (arrays) em Java

- Utilização de vetores

```
int a[] = {1,2,3,4,5,6};

for(int i = 0 ; i< a.length; i++)
{
    System.out.print(a[i] + " ");
}
System.out.println("");

a[0] = 2; //atribui o valor à posição 0 do array

for(int i = 0 ; i<a.length; i++)
{
    System.out.print(a[i] + " ");
}
```

Observação:

Em Java um vetor é um objeto especial que guarda duas coisas:

- Ponteiro para o início do vetor
- Tamanho do vetor!!!!

Portanto já não temos que andar sempre com uma variável às costas, para receber o tamanho do vetor

Para aceder ao tamanho de um vetor guardado numa variável *a*, fazemos:
a.length

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java.exe" ...
1 2 3 4 5 6
2 2 3 4 5 6
Process finished with exit code 0
```

Tipos Estruturados vs Objetos

```c
typedef struct s_square {
    double x; //horizontal coordinate of upper left corner
    double y; //vertical coordinate of upper left corner
    double side; //length of side
} Square;

Square *square(double x, double y, double side){
    Square *s = malloc(sizeof(Square));
    s->x = x;
    s->y = y;
    s->side = side;
    return s;
}
void square_update_position(Square *s, float x, float y){
    s->x = x;
    s->y = y;
}
void square_update_side(Square *s, float side){s->side = side;}
void square_println(Square *s)
{
    printf("Square:(%.2f,%.2f),size:%.2f\n",s->x,s->y,s->side);
}
```

Observação:
Em c, a *struct* define apenas os dados do Tipo Estruturado.

Muitas vezes implementamos funções úteis para trabalhar com o tipo, mas não fazem parte da definição do tipo.

Reparem que as funções (exceto o construtor) costumam receber um ponteiro para o tipo estruturado

Definição do Tipo Estruturado *Square* em C

Tipos Estruturados vs Objetos

- Uma das diferenças entre C e Java, é que Java é uma linguagem Orientada a Objetos
- Em vez de Tipos Estruturados, trabalhamos com Objetos

- Existem várias diferenças entre Tipos Estruturados e Objetos
  - Utilização de Herança
  - Objetos = Dados + Comportamento
  - Modificadores de acesso

Importante:
Quando defino um novo tipo de objeto (classe), defino juntamento com os membros do objeto, um conjunto de funções que estão associadas ao objeto.

Estas funções, só fazem sentido ser invocadas quando associadas a um caso particular de um objeto (designada de instância).

Às funções que estão definidas dentro de um objeto dá-se o nome de métodos.

Objeto Square em Java

```
class Square {
    float x;
    float y;
    float side;

    Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    void updatePosition(float x, float y) {
        this.x = x;
        this.y = y;
    }

    void updateSide(float side) {this.side = side;}
    void println() {
        System.out.println("Square:(" + this.x + "," +
this.y + "),size:"+this.side);
    }
}
```

A primitiva class é usada para definir uma classe/objeto em Java.

membros

Acesso a um membro do objeto, semelhante a C

Métodos definidos dentro da própria classe

Objeto Square em Java

```
class Square {
    float x;
    float y;
    float side;

    Square(float x, float y, float side)
        this.x = x;
        this.y = y;
        this.side = side;
    }

    void updatePosition(float x, float y)
        this.x = x;
        this.y = y;
    }

    void updateSide(float side) {this.side = side;}
    void println() {
        System.out.println("Square:(" + this.x + "," +
this.y + "),size:"+this.side);
    }
}
```

Métodos com o mesmo nome que a classe são métodos especiais designados como construtores.

Um construtor em Java não declara nenhum valor de retorno.

Um construtor em Java apenas tem que inicializar o objeto, pois o objeto é alocado em memória de forma automática pelo Java.

Existe uma variável especial em Java, ***this*** que dentro de um método refere-se sempre ao objeto sobre o qual o método está a ser invocado

Objeto Square em Java

```
class Square {
    float x;
    float y;
    float side;

    Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    void updatePosition(float x, float y) {
        this.x = x;
        this.y = y;
    }

    void updateSide(float side) {this.side = side;}
    void println() {
        System.out.println("Square:(" + this.x + "," +
this.y + "),size:"+this.side);
    }
}
```

membros

Ao contrário da linguagem C, estes métodos não precisam de receber o objeto Square como argumento, pois o Java já nos vai dar acesso ao objeto pretendido (por exemplo através da variável ***this***)

Objeto Square em Java

```
class Square {
    float x;
    float y;
    float side;

    Square(float x, float y, float side)
        this.x = x;
        this.y = y;
        this.side = side;
    }

    void updatePosition(float x, float y)
        this.x = x;
        this.y = y;
    }

    void updateSide(float side) {this.si
    void println() {
        System.out.println("Square:(" + this.x + ","
this.y + "),size:"+this.side);
    }
}
```

membros

Observação:

Também não preciso de incluir o nome do tipo no nome das funções.

Isto era feito em C por várias razões:

- Não podem haver 2 funções com o mesmo nome, e se para além da função update_position para um Square tivesse outra update_position para um Rectangle, tinha que lhes dar nomes diferentes
- Ajuda a agrupar as funções que trabalham com o tipo

Em Java, posso ter métodos com nomes iguais em classes diferentes!

Até posso ter dois métodos com o mesmo nome na mesma classe, desde que tenham argumentos diferentes

Struct vs Class

```c
typedef struct s_square {
    float x;
    float y;
    float side;
} Square;

Square *square(float x, float y, float side){
    Square *s = malloc(sizeof(Square));
    s->x = x;
    s->y = y;
    s->side = side;
    return s;
}
void square_update_position(Square *s, float
x, float y){
    s->x = x;
    s->y = y;
}
void square_update_side(Square *s, float
side){
    s->side = side;
}
void square_println(Square *s)
{
   printf("Square:(%.2f,%.2f),size:%.2f\n",
     s->x,s->y,s->side);
}
```

```java
class Square {
  float x;
  float y;
  float side;

  Square(float x, float y, float side) {
     this.x = x;
     this.y = y;
     this.side = side;
  }

  void updatePosition(float x, float y){
     this.x = x;
     this.y = y;
  }

  void updateSide(float side)
  {this.side = side;}

  void println() {
     System.out.println("Square:(" +
       this.x + "," + this.y +
       "),size:"+this.side);
  }
}
```

UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Struct vs Class

```c
typedef struct s_square {
    float x;
    float y;
    float side;
} Square;

Square *square(float x, float y, float side){
    Square *s = malloc(sizeof(Square));
    s->x = x;
    s->y = y;
    s->side =
    return s;
}
void square_u
x, float y){
    s->x = x;
    s->y = y;
}
void square_update_side(Square *s, float
side){
    s->side = side;
}
void square_println(Square *s)
{
   printf("Square:(%.2f,%.2f),size:%.2f\n",
     s->x,s->y,s->side);
}
```

```java
class Square {
  float x;
  float y;
  float side;

  Square(float x, float y, float side) {
     this.x = x;
     this.y = y;
     this.side = side;
  }

  void updateSide(float side)
  {this.side = side;}

  void println() {
     System.out.println("Square:(" +
       this.x + "," + this.y +
       "),size:"+this.side);
  }
}
```

Embora léxica e sintaticamente diferentes, estas duas implementações são semanticamente equivalentes em C e Java (ou seja, fazem exatamente a mesma coisa)

Utilização de tipos e objetos

```c
int main(void)
{
    Square *s1 = square(1.0,1.0,1.5);
    square_println(s1);
    square_update_position(s1,0.0,0.0);
    square_update_side(s1,2.0);
    square_println(s1);
    printf("%.2f",s1->x);
}
```

Utilização do Tipo Estruturado *Square* em C

```java
public static void main(String[] args)
{
    Square s1 = new Square(1.0f,1.0f,1.5f);
    s1.println();
    s1.updatePosition(0.0f,0.0f);
    s1.updateSide(2.0f);
    s1.println();
    System.out.println(s1.x);
}
```

Utilização da classe *Square* em Java

Utilização de tipos e objetos

```c
int main(void)
{
    Square *s1 = square(1.0,1.0,1.5);
    square_println(s1);
    square_update_position(s1,0.0,0.0);
    square_update_side(s1,2.0);
    square_println(s1);
    printf("%.2f",s1->x);
}
```

Utilização do Tipo Estruturado *Square* em C

```java
public static void main(String[] args)
{
    Square s1 = new Square(1.0f,1.0f,1.5f);
    s1.println();
    s1.updatePosition(0.0f,0.0f);
    s1.updateSide(2.0f);
    s1.println();
    System.out.println(s1.x);
}
```

Utilização da classe *Square* em Java

Criação e utilização de objetos

- Declaração de uma variável da classe
- Utilização do operador *new*
  - Aloca memória no Heap para o objecto
  - Inicializa (chamando o constructor apropriado)
  - Retorna uma referência para o objecto (ponteiro)
- Utilização da variável para invocar métodos

Modificadores de acesso

- Quando definimos uma classe podemos escolher que membros e métodos mostrar e esconder

  Private *– apenas a classe onde o método ou membro foi definido pode ter acesso*

  Protected *– subclasses que herdam da classe também podem ter acesso*

  Public *– membro, método ou classe pode ser acedido/a de qualquer lado*

  Default *– quando não se especifica modificador de acesso. Apenas pode ser acedido dentro da mesma package onde está definido.*

Classe Square c/ modificadores acesso

```
public class Square {
    private float x;
    private float y;
    private float side;

    public Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    //seletores
    public float getX() { return this.x;}
    public float getY() { return this.y;}
    public float getSide() {return this.side;}

    //modificadores
    public void setPosition(float x, float y) {
        this.x = x;
        this.y = y;
    }
    public void setSide(float side) {this.side = side;}
    public void println() {
        System.out.println("Square:(" + this.x + "," + this.y + "),size:"+this.side);
    }
}
```

Estamos a indicar que esta classe pode ser usada/instânciada em qualquer lado

O constructor também é público, senão ninguém conseguia criar novas instâncias de objetos Square

Classe Square c/ modificadores acesso

```
public class Square {
    private float x;
    private float y;
    private float side;

    public Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    //seletores
    public float getX() { return this.x;}
    public float getY() { return this.y;}
    public float getSide() {return this.side;}

    //modificadores
    public void setPosition(float x, float y) {
        this.x = x;
        this.y = y;
    }
    public void setSide(float side) {this.side = side;}
    public void println() {
        System.out.println("Square:(" + this.x + "," + this.y + "),size:"+this.side);
    }
}
```

Estamos a indicar que estes membros não podem ser referenciados diretamente fora desta classe

Classe Square c/ modificadores acesso

```
public class Square {
    private float x;
    private float y;
    private float side;

    public Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    //seletores
    public float getX() { return this.x;}
    public float getY() { return this.y;}
    public float getSide() {return this.side;}

    //modificadores
    public void setPosition(float x, float y) {
        this.x = x;
        this.y = y;
    }
    public void setSide(float side) {this.side = side;}
    public void println() {
        System.out.println("Square:(" + this.x + "," + this.y + "),size:"+this.side);
    }
}
```

Seletores (ou getters) são métodos usados quando se pretende dar acesso a membros privados da classe. Neste caso declaramos estes seletores como públicos.

Classe Square c/ modificadores acesso

```
public class Square {
    private float x;
    private float y;
    private float side;

    public Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    //seletores
    public float getX() { return this.x;}
    public float getY() { return this.y;}
    public float getSide() {return this.side;}

    //modificadores
    public void setPosition(float x, float y) {
        this.x = x;
        this.y = y;
    }
    public void setSide(float side) {this.side = side;}
    public void println() {
        System.out.println("Square:(" + this.x + "," + this.y + "),size:"+this.side);
    }
}
```

Modificadores (ou getters) são métodos usados quando se pretende dar acesso de escrita a membros privados da classe. Neste caso declaramos estes modificadores como públicos.

Classe Square c/modificadores acesso

- Pq usar modificadores e seletores se posso simplesmente dar acesso público aos membros da classe?

- A utilização de modificadores e seletores permite-nos muito maior controle sobre a forma como os membros são acedidos e alterados.
  - Ex: podemos evitar que alguém altere de forma incorreta um membro.
  - Podemos não querer dar acesso a alguns membros, ou dar acesso parcial.

Classe Square apenas de leitura

Observação:
Esta versão da classe Square, que não tem modificadores, implementa quadrados só de leitura. Ou seja, uma vez criado um novo *Square*, não é possível alterar a posição ou largura desse quadrado.

Em C, não é possível criar uma struct com esta restrição

```
public class Square {
    private float x;
    private float y;
    private float side;

    public Square(float x, float y, float side) {
        this.x = x;
        this.y = y;
        this.side = side;
    }

    //seletores
    public float getX() { return this.x;}
    public float getY() { return this.y;}
    public float getSide() {return this.side;}

    public void println() {
        System.out.println("Square:(" + this.x + "," + this.y + "),size:"+this.side);
    }
}
```

Objetos e Herança

```
public class ContaBancaria {
	protected long NIB;
	protected double saldo;

	public ContaBancaria(long nib)
	{
		NIB = nib;
		saldo = 0;
	}
	public double levantamento(double valor)
	{
		if(saldo >= valor)
		{
			//levantamento efectuado com sucesso
			saldo -= valor;
		}
		return saldo;
	}
	public double deposito(double valor)
	{
		if(valor > 0)
		{
			saldo += valor;
		}
		return saldo;
	}
}
```

```
public class Main {

	public static void main(String[] args)
	{
		ContaBancaria conta = new ContaBancaria(2923948);
		conta.deposito(10.0);
	}
}
```

Herança

- Permite a definição de uma nova classe a partir de uma classe já existente
- Favorece a reutilização de código

```
public class ContaCartaoCredito extends ContaBancaria {

	private double credito;

	public ContaCartaoCredito(long nib)
	{
		super(nib);
		credito = 2500;
	}

	double levantamentoCredito(double valor)
	{
		if(valor <= credito)
		{
			credito -= valor;
		}

		return credito;
	}
}
```

Observação:
Indicamos que a classe ContaCartaoCredito herda da classe ContaBancária

A classe ContaCartaoCredito vai herdar todos os membros e métodos da classe mãe.

Para além dos membros e métodos herdados podemos definir novos membros e novos métodos.