Programação em Java – Uma Revisão

Referências

- M.T. Goodrich, R. Tamassia. Estruturas de Dados e Algoritmos em Java. 4a Edição. Ed. Bookman
  - Cap. 1
- Tutoriais:
  - http://docs.oracle.com/javase/tutorial/

Linguagens & Orientação a Objetos

- Atualmente, existem diversas linguagens que de alguma forma se baseiam nos princípios de orientação a objetos:
  - Smalltalk
  - C++
  - Objective C
  - Object Pascal (Delphi)
  - Eiffel
  - Java

A Linguagem Java

- Originalmente concebida para a programação de dispositivos eletrônicos (Sun Microsystems)
- Posteriormente utilizada para a programação na Web (páginas de conteúdo dinâmico)
- Atualmente:
  - desenvolvimento de aplicações corporativas em larga escala
  - novas funcionalidades em servidores Web (e.g. servelets)
  - programação de dispositivos móveis (celulares, PDAs)

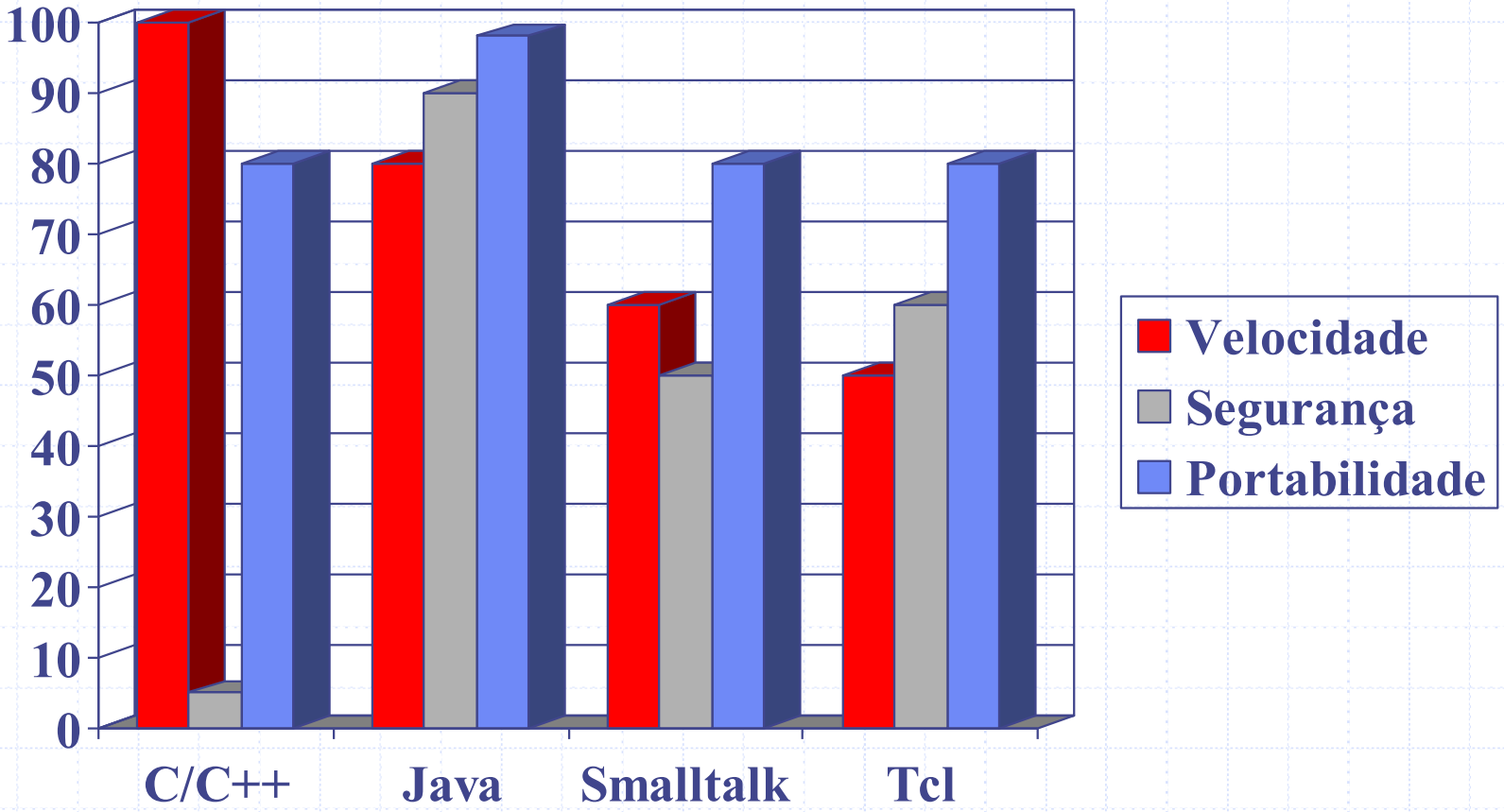
Principais Características

- Fortemente baseada no paradigma de orientação a objetos
- Herdou diversas características de C++ (que por sua vez, herdou de C, ...)
- Linguagem interpretada
- Alta portabilidade
- Executa sobre uma máquina virtual
- Código gerenciável

Principais Características (cont.)

- *Garbage collection* automatizado
- Sintaxe simples (mais simples que C++)
- Biblioteca bastante completa:
  - programação em rede
  - suporte para WWW
  - suporte gráfico
  - diversos algoritmos e estruturas de dados
- Estimula boas práticas de programação

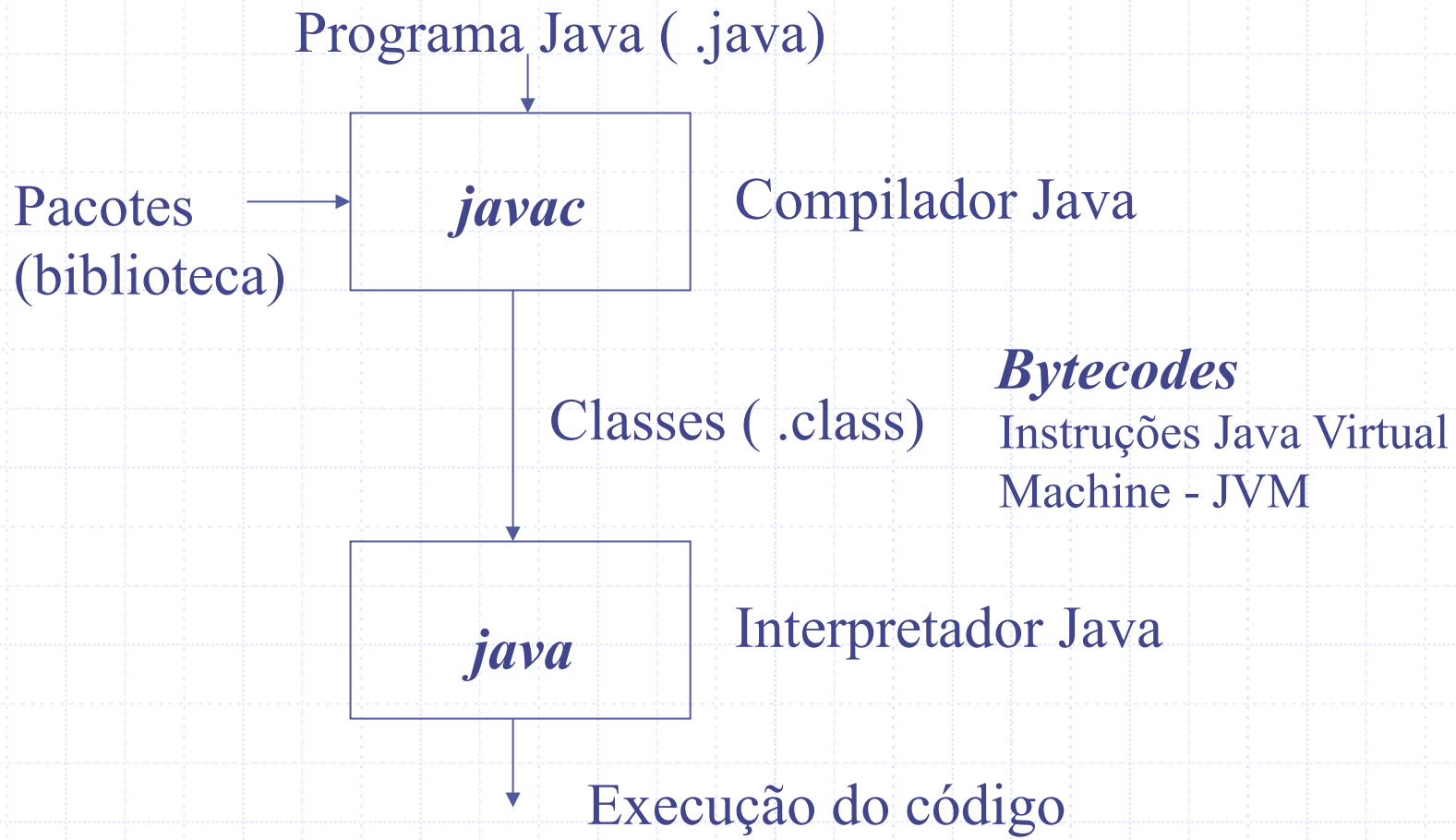
Análise Comparativa



Ferramentas

- Disponível em três plataformas:
  - J2EE – Java 6 Platform, Enterprise Edition
  - J2SE – Java 6 Platform, Standard Edition
  - J2ME – Java 6 Platform, Micro Edition
- Opcionalmente, pode-se instalar somente a máquina virtual (permite execução de aplicações).
- Diversos IDEs (Integrated Development Environment):
  - Eclipse
  - NetBeans
  - JBuilder
  - Jedit
  - Gel
  - OptimalJ
  - Dezenas de outros ...

Fluxo de Trabalho



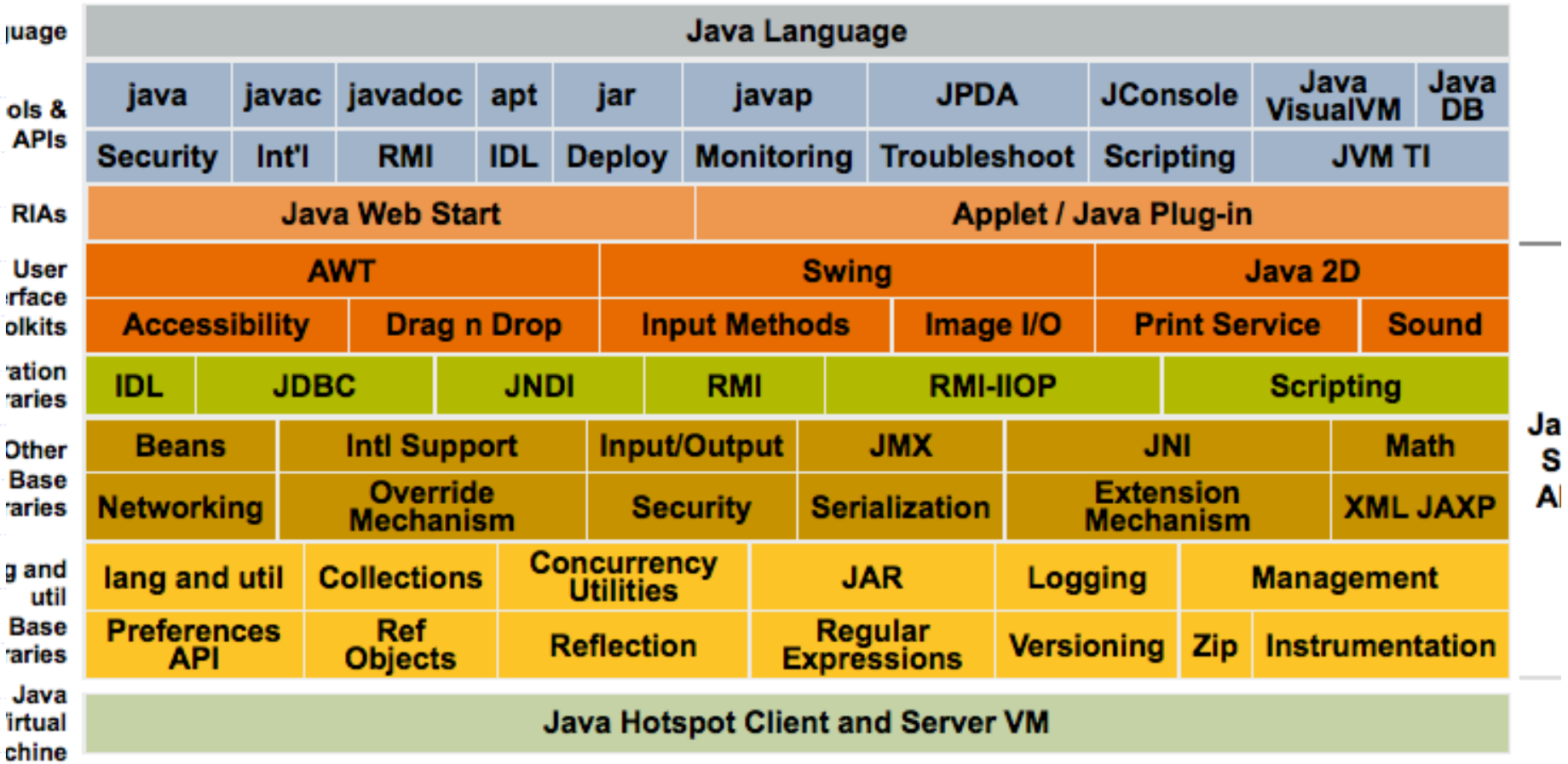
Fases de um Programa

1. O programa (**.**java) é escrito com auxílio de um editor de textos simples ou IDE
2. Programa é compilado (*javac*), gerando o *.class*
3. Para a execução (*java*), o ***classloader*** lê o arquivo *.class* contendo os ***bytecodes*** e carrega em memória
4. Um programa chamado ***bytecodes verifier*** verifica se os *bytecodes* lidos são válidos, e se não violam nenhuma restrição de segurança

Biblioteca

- Java possui uma enorme quantidade de classes implementadas e organizadas em Pacotes:
  - Interface com páginas WWW (applets)
  - Estruturas de dados básicas (pilhas, vetores, hashing)
  - Tratamento de Entrada/Saída (arquivos e impressoras)
  - Interfaces Gráficas (janelas, botões, diálogos)
  - Computação Gráfica (animação)
  - Redes (sockets)
  - Bancos de Dados (conexão, SQL)
  - Acesso Remoto (RMI, CORBA)
  - Processamento de Imagens (filtros, estruturas de dados)
  - Muitos outros ...

Overview


Java Language
java
javac
javadoc
apt
jar
javap
JPDA
JConsole
Java VisualVM
Java DB
Security
Int'l
RMI
IDL
Deploy
Monitoring
Troubleshoot
Scripting
JVM TI
Java Web Start
Applet / Java Plug-in
AWT
Swing
Java 2D
Accessibility
Drag n Drop
Input Methods
Image I/O
Print Service
Sound
IDL
JDBC
JNDI
RMI
RMI-IIOP
Scripting
Beans
Intl Support
Input/Output
JMX
JNI
Math
Networking
Override Mechanism
Security
Serialization
Extension Mechanism
XML JAXP
lang and util
Collections
Concurrency Utilities
JAR
Logging
Management
Preferences API
Ref Objects
Reflection
Regular Expressions
Versioning
Zip
Instrumentation
Java Hotspot Client and Server VM

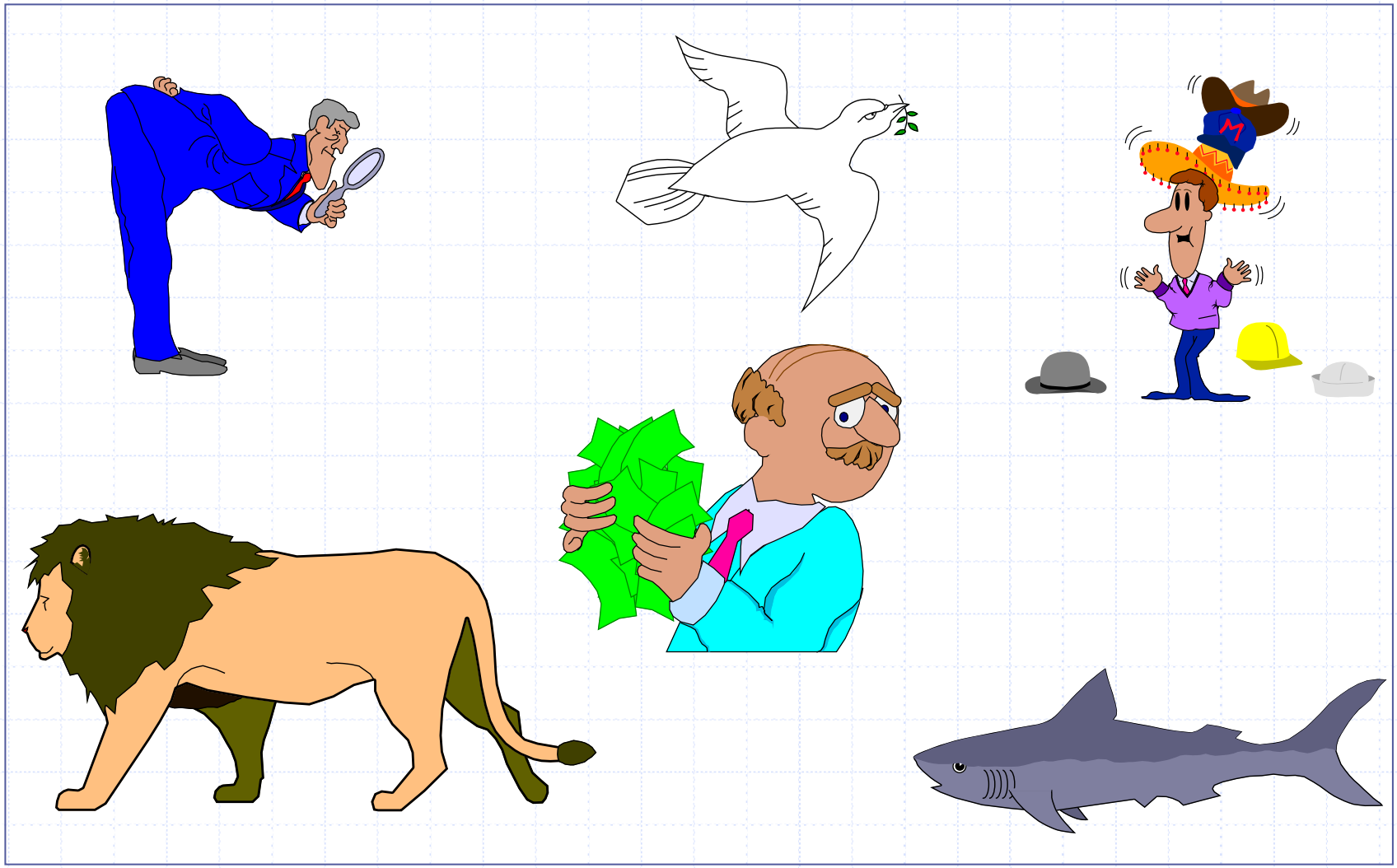
Palavras-Chave

Abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
final	finally	float	for	goto
if	implements	import	instanceof	int
interface	long	native	new	null
package	private	protected	public	return
short	static	super	switch	
synchronized	this	throw	throws	transient
try	void	volatile	while	

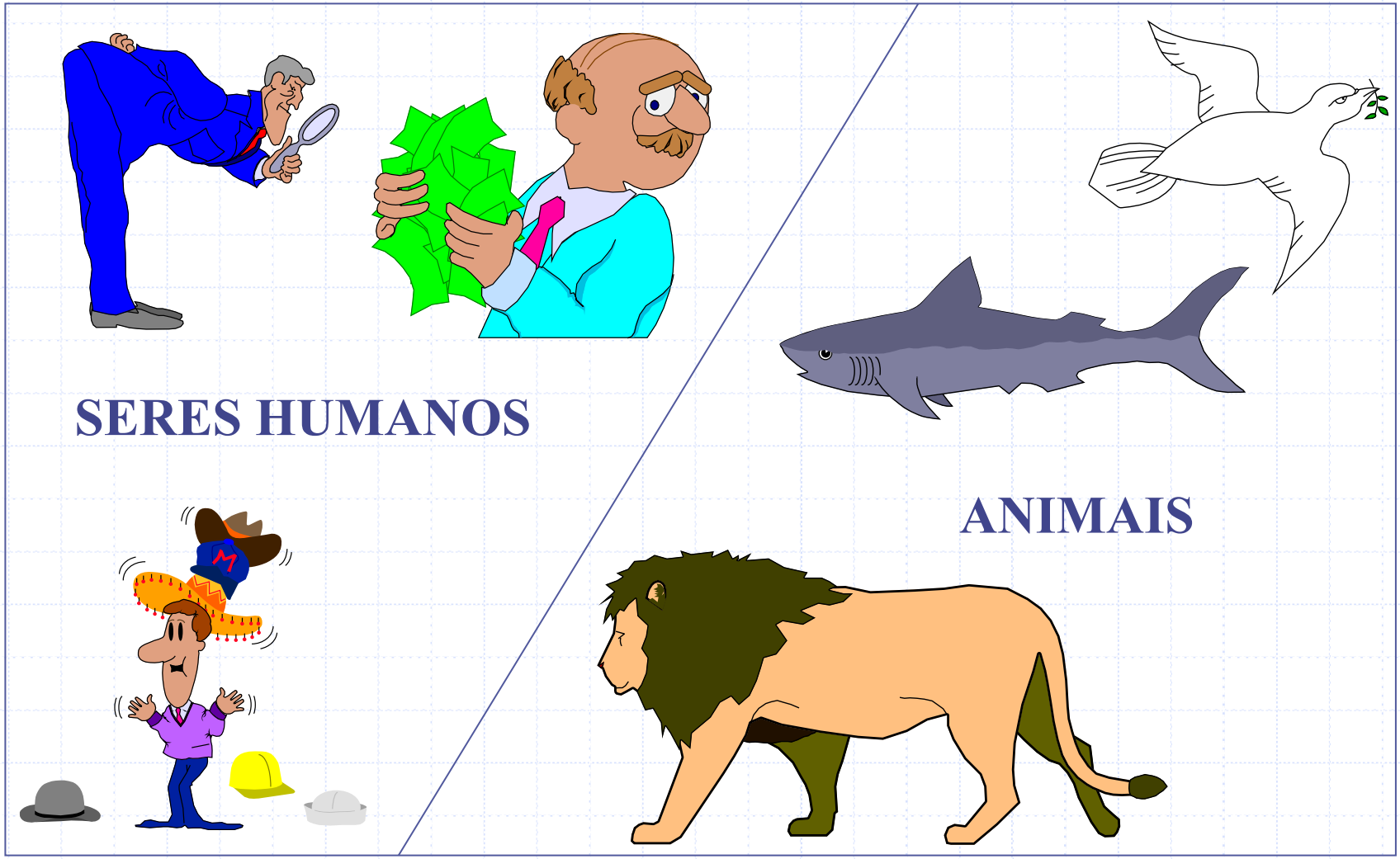
Classes e Objetos

- Classes definem as características de um conjunto de objetos.
  - Usadas para a modelagem de entidades do mundo real. Ex.: Pessoas, animais, produtos, etc
- Em programação orientada a objetos (OOP) as classes permitem a declaração de dois tipos de características:
  - os **atributos** são dados relativos a cada objeto
  - os **métodos**, que implementam as operações que podem atuar sobre os atributos

Objetos

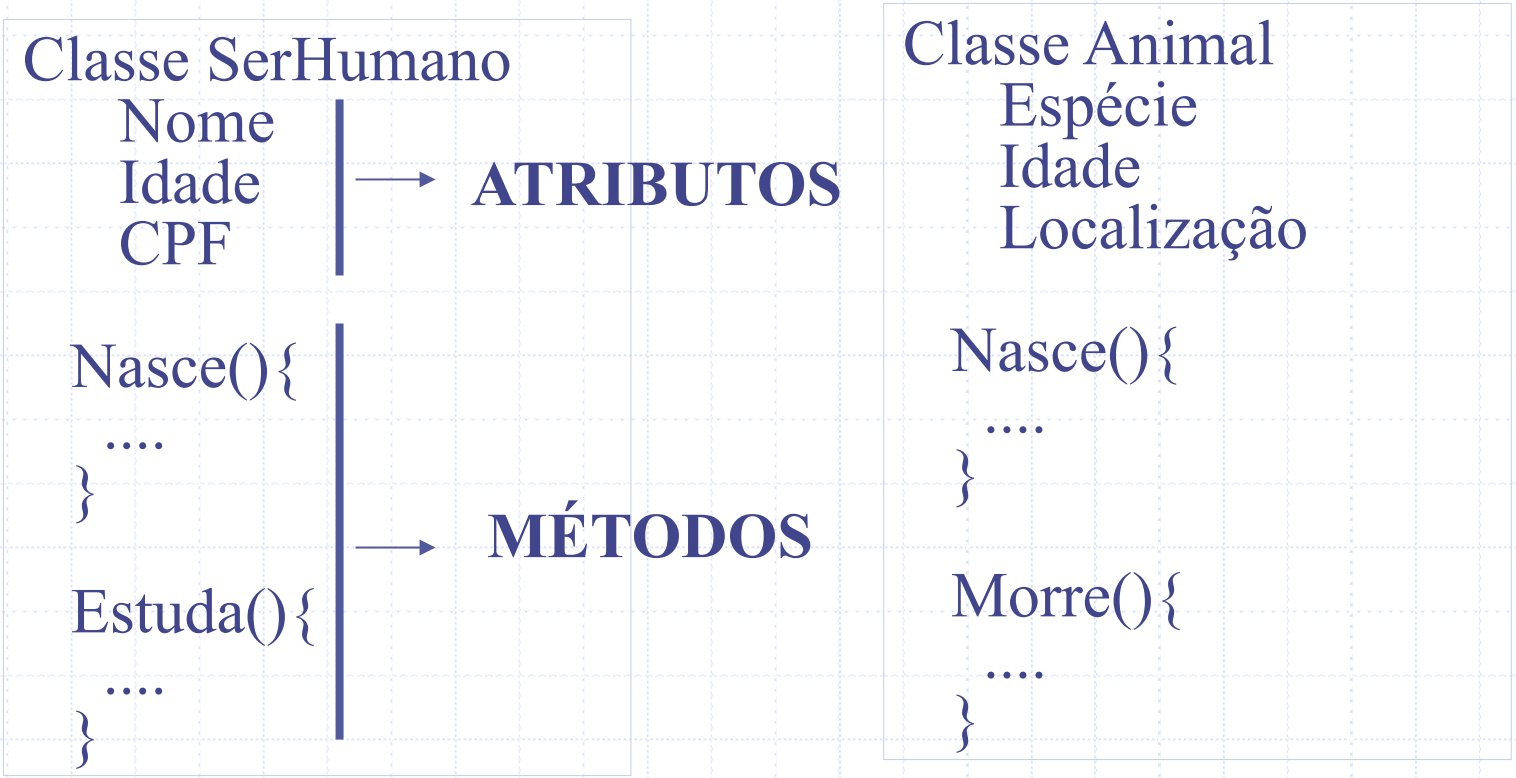


Classificando os Objetos



Classes

- São abstrações de conjuntos de objetos:



Instanciação

- Instância = objeto da classe

Classificação

```
class SerHumano
      Nome
      Idade
      CPF

   Nasce(){
      ....
   }

   Estuda(){
      ....
   }
```

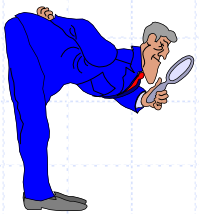
Instanciação

```
Objeto OBJ1
      Nome=Mr.Jones
      Idade=34

CPF=4567889484

   Nasce(){
      ....
   }

   Estuda(){
      ....
   }
```



```
Objeto OBJ2
      Nome=Mr.Zoo
      Idade=45

CPF=6786968696

   Nasce(){
      ....
   }

   Estuda(){
      ....
   }
```

Declaração de Classes

[<modificadores> **class** < nome da classe >**{**

[extends <nome da superclasse>**]**

[implements <nome da interface1>,

<nome da interface2>, ...**]**

// declarações dos atributos e métodos

}

Modificadores de Classes

- O modificador **abstract** indica que uma classe possui um ou mais métodos abstratos, isto é, que são declarados mas não são implementados por ela (apenas pelas subclasses)
- O modificador **final** indica que a classe não admite subclasses
- O modificador **public** indica que a classe pode ser utilizada (i.e., instanciada ou estendida) por qualquer objeto/classe que esteja no mesmo pacote ou que a importe:
  - classes **public** só podem ser declaradas em um arquivo com o mesmo nome da classe (i.e., <nome_da_classe>.java)
  - só pode haver uma classe pública por arquivo fonte em Java
- Caso contrário, a classe é considerada ***amigável***, por convenção, o que significa que pode ser utilizada por qualquer objeto/classe que esteja no **mesmo pacote**. Será pública caso a classe não esteja em algum pacote.

Atributos

- Os atributos são também chamados de **variáveis de instância** por armazenarem dados particulares sobre uma instância
- Atributos podem ter os seguintes modificadores:
  - **public**: livre acesso
  - **protected**: somente métodos de subclasses ou classes do mesmo pacote podem acessar
  - **private**: somente métodos da própria classe podem acessar. Nem mesmo as subclasses têm acesso.
  - se nenhum modificador for declarado, então o atributo é automaticamente *amigável* por convenção.

Atributos

- Além dos modificadores que operam sobre a visibilidade dos atributos, existem também:
  - **static**: indica que esse é um atributo da classe, e não da instância (i.e., o valor desse atributo é o mesmo para todas as instâncias da classe).
  - **final**: determina que o atributo deve receber um valor inicial, que não pode ser modificado no futuro.

Tipos Básicos

- Atributos podem ser instâncias de outras classes (objetos), ou podem ser de tipos básicos:

Tipo	Tam.	Descrição
boolean	1 bit	valor lógico **true** ou **false**
char	16 bits	caracter em formato Unicode (alfabetos latino, grego, cirílico, ideogramas japonês, etc)
byte	8 bits	um inteiro com sinal: de **-128 a 127**
short	16 bits	um inteiro com sinal: de **-32768 a 32767**
int	32 bits	um inteiro com sinal: **-2.147.483.648 a 2.147.483.647**
long	64 bits	inteiro longo com sinal: de **-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807**
float	32 bits	número de ponto flutuante em formato IEEE 754: de **-3,4E-38... +3,4E+38**
double	64 bits	número de ponto flutuante em formato IEEE 754: de **:-1,7E-308... +1,7E+308**

Objetos Numéricos

- Algumas vezes pode ser interessante armazenar números como objetos. Para isso existem as classes numéricas:

Tipo	Nome da Classe	Exemplo de criação	Exemplo de Acesso
byte	`Byte`	`n=new Byte((byte)34)`	`n.byteValue()`
short	`Short`	`n=new Short((short)99)`	`n.shortValue()`
int	`Integer`	`n=new Integer((int)734)`	`n.intValue()`
long	`Long`	`n=new Long(1067L)`	`n.longValue()`
float	`Float`	`n=new Float(5.93F)`	`n.floatValue()`
double	`Double`	`n=new Double(5.93)`	`n.doubleValue()`

Exemplo

```
class Gnomo{
  // atributos
  protected    String    nome;
  protected    int       idade;
  protected    Gnomo          amigo;
  private      boolean   magico;
  public       double    altura=0.75;
  public  static final int ALT_MAX = 0.95;
  // definições de métodos ...
}
```

Métodos

- Descrevem as operações efetuadas sobre os atributos
- São trechos de código que podem ser chamados por outros métodos ou pelo programa principal (*main*)
- São semelhantes às funções ou subrotinas da programação procedural tradicional
- Podem receber parâmetros
- Podem devolver resultados

Declaração de Métodos

[<modificadores>] < tipo de retorno > <nome do metodo> ([<parametros>]){

    // corpo do método

}

Modificadores de Métodos

- Os métodos, assim como os atributos, também podem ter sua visibilidade definida pelos modificadores
  - **public**: chamada livre
  - **protected**: somente métodos de subclasses ou classes do mesmo pacote podem chamar
  - **private**: somente métodos da própria classe podem chamar. Nem mesmo as subclasses têm acesso.
  - se nenhum modificador for declarado, então o método é ***amigável*** por convenção.

Modificadores de Métodos (cont.)

- Além dos modificadores de visibilidade, existem também:
  - **static**: indica que esse método esta associado à classe, e não a uma instância, i.e., pode-se chamá-lo mesmo sem ter uma instância (e.g. para modificar um atributo *static*)
  - **final**: este método não pode ser rescrito por qualquer subclasse
  - **abstract**: significa que o método não tem código (só podem ocorrer em classes abstratas)

Métodos Construtores

- São métodos especiais, que são invocados implicitamente quando uma nova instância é criada. É usado geralmente para inicializar objetos. Ex.:

```
public Gnomo (String n) {
        nome = n;
        idade = 150;
        magico = true;
}
```

- Quando qualquer outro construtor é declarado, o **construtor padrão** não existe mais e pode ser definido explicitamente caso desejado (e.g. para evitar erro na chamada implícita desse construtor feita quando da instanciação de uma subclasse) .

Operadores

- Java possui quatro tipos de operadores:
  - básicos
  - aritméticos
  - relacionais
  - lógicos e binários

Operadores Básicos

.	(ponto) referência a método, função ou atributo de um objeto
,	separador de identificadores
;	separador de declarações e comandos
[ ]	declarador de matrizes e delimitador de índices
{ }	separador de blocos e escopos locais
( )	precedência de operadores, listas de parâmetros

Operadores Aritméticos

+	adição	x = x + 6
-	subtração	x = x - 6
*	multiplicação	x = x * 6
/	(quociente inteiro da) divisão	x = y / 4
%	resto inteiro da divisão (inteira)	x = y % 5
++	incremento de 1 (inteiros)	x++ (pós-incremento)
		++x (pré-incremento)
--	decremento de 1 (inteiros)	x--
		--x

Operadores Aritméticos (cont.)

+=	atribuição aditiva	x+=3
	Tem o mesmo efeito de	x=x+3
-=	atribuição subtrativa	x-=3
*=	atribuição multiplicativa	x*=3
/=	atribuição divisiva	x/=3
%=	atribuição de resto	x%=3

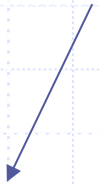
PS. Potência ⇒ Método "pow" da classe "Math" do pacote "java.lang"

Operadores Relacionais

==	igual	x == y
!=	diferente	x != y
>	maior que	x > y
<	menor que	x < y
>=	maior ou igual	x >= y
<=	menor ou igual	x <= y

Operadores Binários

~	complementação de bits	~x
>>	*shift right*	x>>3
>>>	*shift right* sem sinal	x>>>3
<<	*shift left*	x<<3
>>=	atribuição com *shift right*	x>>=3
<<=	atribuição com *shift left*	x<<=3
>>>=	atribuição com *shift right* sem sinal	x>>>=3



No de deslocamentos

Operadores Lógicos e Binários

&	and lógico bit a bit	x&y
|	or lógico bit a bit	x|y
^	xor lógico bit a bit	x^y
&=	atribuição com &	x&=y
|=	atribuição com |	x|=y
^=	atribuição com ^	x^=y
? :	*if-then-else* lógico	x=((y>3)?5:6)
||	*or* condicional	(x>y) || (z>=6)
&&	*and* condicional	(x>y) && (z<4)

São *dinâmicos* no sentido que não avaliam o 2o operando se não for necessário

Precedência de Operadores

Aumenta prioridade

```
( ) [] . ; ,
++ -- !
* / %
+ -
>> << >>>
> <  >=  <=
== !=
&
^
|
&&
||
?:
=
```

Diminui prioridade

Estrutura de Controle Condicional

- Formato:

```
if ( expressão booleana )
        bloco 1;
else bloco2;
```

Condicional

Exemplo de controle condicional

```
if ((x>5) && (y!=70)){
        x=x+5;
        y=z+48;
}
else if (x<=5)
                x=48;
        else x=20;
```

Seleção

```
switch (expressão){
      case valor1: bloco1;
                   break;
        ...
      case valorN: blocoN;
                   break;
     default: bloco N+1;
 }
```

Seleção (exemplo)

```
clicado = ...
switch(clicado){
        case 0: saida=3;
                break;
        case 1: saida=2;
                break;
        default: saida=0;
}
System.out.println(saida);
```

Repetição

- Repetição com laço *while*

```
while( expressão booleana ){
    bloco;
}
```

Exemplo de Laço com *while*

```
while (x > 6 ) {
        while ( y > 9 ){
                while((z>=6) && (m!=10)){
                        z++;
                        m%=10;
                }
                y=z+m;
        }
        x+=y;
}
```

Repetição com laço *do-while*

```
do{
    Bloco;
} while(expressão booleana);
```

Exemplo de laço com *do-while*

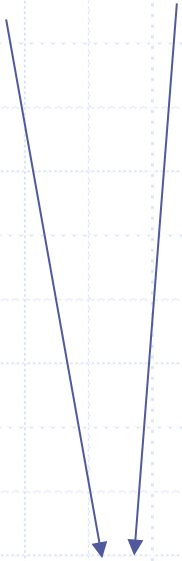
```
do{
        do{
                do{
                        z++;
                        m%=10;
                } while((z>=6) && (m!=10));
                y=z+m;
        } while(y>9);
        x+=y;
} while(x>6);
```

Repetição com laço *for*

```
for(inicialização;condição-limite;iteração){
        Bloco;
}
```

Exemplo de laço com *for*

```
for(x=0,y=0;(x<100) & (y<=20));x++,y++){
        for (int z=-5;z<10;z+=3)
                System.out.println(z);
        System.out.println(y);
  }
```



Incremento Simultâneo

Declaração de *arrays*

tipo <variável>[ ];
tipo [ ]<variável>;
tipo <variável>[ ][ ];
tipo [ ][ ] <variável>;

...

tipo <variável>[ ]=new tipo[dimensão];
tipo <variável>[ ][ ]=new tipo[dim1][dim2];

Exemplos de Declaração de *arrays*

```
int vetor[ ];
int [ ]vetor;
float matriz[ ][ ];
boolean [ ][ ]m;

int vetor[ ] = new int[100];
boolean matriz[ ][ ] = new boolean[100][100];
```

Inicialização de *arrays* na Declaração

```
int v[ ]={3,4,5};
float matriz[ ][ ]={{1.5,1.6,1.9},
                    {1.5*2,3,6},
                    {5*(1.2+2),8,9}
                   }
```

Referências a Elementos dos *arrays*

```
for(i=0;i<100;i++)
        v[i]=i*40;
        v[0]=v[0]-100;
for(i=1;i<50;i++)
        m[i][i]=v[i]-v[0];
```

Padrões de Comentário

- /* .....*/ (Estilo C)
- // ..... (Estilo C++)
- Usar marcador /** bem como comandos especiais (e.g. @author, @version) e aplicar o programa *javadoc* para gerar a documentação da API (Application Program Interface) referente a uma classe ou pacote no padrão html

Referências

- Download:
  - http://www.java.com/en/download/manual.jsp
- Tutoriais:
  - http://docs.oracle.com/javase/tutorial/
- Documentação Online:
  - http://docs.oracle.com/javase/6/docs/api/
  - http://docs.oracle.com/javase/6/docs/
- Versão para download:
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html#docs