

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA - CCET
CURSO DE CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA DE COMPILADORES I

A Linguagem

JAVA

Henry Franklin Duailibe da Costa (CP 99106-22)

Igor de Jesus Mesquita (CP 98204-19)

São Luís/MA

jan. de 2001

SUMÁRIO

1. Histórico da Linguagem.....	4
1.1. O Projeto Green.....	5
1.2. WebRunner e HotJava.....	5
1.3. Garbage Collection (Libera uma coleção).....	6
2. Adquirindo o software necessário e plataformas suportadas.....	6
3. Características da linguagem	7
3.1. Parecida com C, C++.....	7
3.2. Compilada	7
3.3. Portável.....	7
3.4. Orientada a Objetos	8
3.5. Segura	8
3.6. Suporta concorrência	9
3.7. Eficiente	9
3.8. Suporte para programação de sistemas distribuídos	9
4. Tipos de Dados	10
4.1. Comentários	10
4.2. Ponto e vírgula, Blocos e o espaço em branco.....	10
4.3. Identificadores	10
4.4. Tipos Básicos no Java.....	10
4.4.1. Tipo Lógico.....	10
4.4.2. Tipo Textual	10
4.5. Tipo Ponto Flutuante	11
5. Tipos de dados avançados:	11
5.1. Declaração de Arrays	11
5.1.1. Criando um Array	11
5.1.2. Arrays Multi-dimensionais.....	12
6. Variáveis e Tempo de vida	12
6.1. Inicialização de variáveis	12
7. Operadores.....	13
7.1. Concatenação	14
7.2. Casting (Conversão de tipos).....	14
8. Instruções de controle:	14
9. Palavras Reservadas.....	15
10. Estruturação teórica da linguagem:	16
10.1. Classes e objetos:	16
10.10. Subclasses (Relacionadas).....	18
10.11. Subclasses (Extensões).....	18

10.12. Herança Simples	19
10.13. Polimorfismo	19
10.14. Argumentos de Métodos e Coleções Heterogêneas.....	19
10.15. O Operador instanceof.....	20
10.16. Objetos Lançados	20
10.17. Sobre-escrevendo Métodos	20
10.2. Abstração de Tipos de Dados	16
10.3. Definição de Métodos	16
10.4. Passagem de Valores	17
10.5. A Referência This	17
10.6. Ocultando Dados	17
10.7. Encapsulamento.....	17
10.8. Sobrescrevendo Métodos	18
10.9. Construtores.....	18
11. Comparando Java com outras Linguagens.....	21
11.1. Classes se agrupando - Pacotes	21
11.2. A declaração import	22
12. O que é um Applet?	22
12.1. Carregando um Applet	22
12.2. Segurança e Restrições nos Applet	22
12.3. Métodos de Applet chaves.....	22
12.4. Applet Display	23
13. Ponteiros, “Pointers”, Referências e Objetos.....	23
13.1. Passagem por referência	23
13.2. Vetores e matrizes.....	23
14. Alocação Dinâmica	25
14.1. Representação linear de uma matriz	25
14.2. Matrizes definidas na linguagem:	26
15. Conclusão (Visão geral da linguagem):	26

1. Histórico da Linguagem

Java é uma linguagem de programação ideal para aplicativos para a Internet. Mostraremos a seguir as origens da linguagem, e porque ela é tão apropriada para ser usada na Internet, particularmente na WorldWide Web.

Embora as applets Java sejam escritas na linguagem Java, não é necessário saber programar em Java para usá-las. Se você está interessado em usar applets sem realmente programar em Java, pode pular esta parte, que introduz a linguagem.

Você pode ler a definição da linguagem Java (em inglês), no site Java/HotJava da Web. <http://www.javasoft.com/hooks/language-ref.html> ***A História da Linguagem Java*** A história de uma linguagem de programação pode contar muita coisa sobre a linguagem em si. Durante a fase de projeto, é importante usar a linguagem em aplicativos reais. Do contrário, a linguagem não será tão útil quando for lançada, e não será tão divertido programar nela.

A linguagem de programação Java foi usada no desenvolvimento de diversos aplicativos enquanto estava sendo projetada. Originalmente, pretendia-se que a Java fosse usada na programação de produtos eletrônicos de consumo (eletrodomésticos, eletroeletrônicos). Entretanto, ela se transformou em uma grande linguagem para programação para a WorldWide Web.

As Origens da Java A linguagem de programação Java foi projetada e implementada por uma pequena equipe de pessoas coordenada por James Gosling na Sun Microsystems em Mountain View, Califórnia. Além de seu trabalho com a Java, James Gosling é o autor do emacs do Unix e do sistema de janelas neWs. <http://www.javasoft.com/people/jag/index.html>. A equipe Java original trabalhava no projeto de software para produtos eletrônicos de consumo. Rapidamente, eles descobriram que as linguagens de programação existentes, como C e C++, não eram adequadas.

Programas escritos em C e C++ têm de ser compilados para cada tipo específico de processador (chip de CPU). Quando um novo processador é lançado, a maior parte do software tem de ser recompilada, para aproveitar os novos recursos do processador. Depois de compilados, os programas em C e C++ não são facilmente adaptáveis ao uso de novas bibliotecas de software. Os programas têm de ser totalmente recompilados quando a biblioteca muda.

Entretanto, softwares para produtos de eletrônica de consumo devem funcionar em novos processadores, porque os fabricantes têm como restrição o custo dos componentes. Se o preço de um processador fica muito alto, eles o substituirão imediatamente por um mais novo, de custo-benefício mais atraente. Mesmo pequenas variações de preço podem fazer diferença, quando se pensa em termos de vendas de milhões de unidades.

Outro problema no uso de linguagens de programação tradicionais para o software de produtos eletrônicos de consumo está em que o consumidor quer uma vida útil longa para seu produto. Há torradeiras funcionando que têm 50 anos de idade. O plugue ainda se encaixa na tomada e as fatias de pão ainda se encaixam nas aberturas. Os softwares normalmente tem uma vida útil bem mais curta, o que tornaria difícil construir uma torradeira com um computador embutido. Sempre que novos programas para torradeiras fossem escritos, eles teriam de apresentar compatibilidade retroativa, pois o software das torradeiras antigas teria de ser substituído.

O software usado em produtos eletrônicos de consumo também precisa ser muito confiável, muito mais do que o software para computadores. Se um eletrodoméstico ou eletroeletrônico falha, o fabricante geralmente tem de substituir todo o aparelho.

Em 1990, James Gosling começou a projetar uma linguagem de programação nova, que seria mais apropriada para produtos eletrônicos de consumo, sem os problemas de linguagens tradicionais como C e C++. O resultado é Java, uma linguagem muito rápida, pequena e confiável, que rodará em todos os tipos de processadores.

1.1. O Projeto Green

O primeiro projeto a usar a linguagem Java foi o projeto Green. Seu propósito era testar um novo tipo de interface do usuário para controlar um ambiente doméstico (videocassete, televisor, luzes, telefone, e assim por diante). As pessoas que trabalhavam no projeto Green construíram um computador experimental de mão, que recebeu o nome-código de *7 (pronuncia-se "star seven").

A interface do usuário consistia em uma representação animada em cores da casa, na qual os aparelhos eletrodomésticos podiam ser manipulados tocando-se na tela. Ela era, é claro, completamente escrita em Java. Oito protótipos operacionais do *7 foram construídos.

A interface do usuário *7 usava figuras animadas na tela para controlar os eletrodomésticos. A equipe de desenvolvimento Java ainda usa algumas das figuras do projeto *7. Duke, que agora é o mascote da Java, foi uma figura animada neste projeto. O projeto que deu continuidade ao *7 foi uma demonstração de vídeo-por-demanda (VPD). Este projeto demonstrou que a interface do usuário com figuras de desenho animado do *7 poderia ser usada para TV interativa da mesma forma que para controlar eletrodomésticos.

Os projetos *7 e VPD resultaram em software experimental, mas não se transformaram em produtos reais. Ambos os projetos foram implementados inteiramente em Java e ajudaram a amadurecer a linguagem Java.

1.2. WebRunner e HotJava

Na época em que Arthur e Sami entraram no projeto Java, em 1993, a WorldWide Web estava passando de uma interface baseada em texto para uma mais gráfica, gerando assim, enorme interesse. Ocorreu então à equipe de desenvolvimento Java que uma linguagem independente de plataforma como a Java seria ideal para a programação de aplicativos Web, uma vez que um programa em Java poderia rodar nos diversos tipos de computadores ligados à Internet, de PCs a Macs e sistemas Unix. O resultado foi um browser da Web chamado WebRunner inteiramente escrito em Java. Mais tarde, por razões comerciais, este browser foi rebatizado de HotJava, e foi o primeiro suporte a applets Java.

O HotJava foi um passo importante para a linguagem Java. Ele não apenas amadureceu a linguagem, mas também apresentou-a ao mundo. Quando outros programadores viram o que a equipe de desenvolvimento Java estava fazendo com o HotJava, muitos deles quiseram usar esta nova tecnologia em seus produtos de software, também.

O anúncio oficial da tecnologia Java foi feito em maio de 1995 na conferência SunWorld em San Francisco (EUA). Nela, Marc Andreessen, um dos fundadores e vice-presidente de tecnologia da Netscape Communications, anunciou que o Netscape Navigator daria suporte a applets Java. O Netscape Navigator 2.0, lançado no fim de 1995, dá suporte à Java e aumentou ainda mais o interesse já existente pela tecnologia Java. A equipe de desenvolvimento Java, localizada em Palo Alto, Califórnia, continua a refinar e a desenvolver a linguagem e o browser HotJava para corresponder a este interesse . <http://www.javasoft.com/hooked/people.html>

1.3. Garbage Collection (Libera uma coleção)

O Java não segura áreas de memória que não estão sendo utilizadas, isto porque ele tem uma alocação dinâmica de memória em tempo de execução.

No C e C++ (e em outras linguagens) o programa desenvolvido é responsável pela alocação e desalocação da memória.

Durante o ciclo de execução do programa o Java verifica se as variáveis de memória estão sendo utilizadas, caso não estejam o Java libera automaticamente esta área que não esta sendo utilizada.

2. Adquirindo o software necessário e plataformas suportadas

Um ambiente de programação Java é normalmente composto de um kit de desenvolvimento de aplicações Java e um “browser compatível com esta linguagem (recomendável). Se você não tem acesso a esse ambiente de programação, tente estes endereços:


“DOWNLOAD”
JAVA

<http://Java.sun.com>

Raiz do hipertexto montado pelos criadores da linguagem. Sob este endereço você pode obter o compilador e outras ferramentas de desenvolvimento de aplicações Java para a sua plataforma de programação. Fique atento! Outros desenvolvedores estão criando ambientes de programação Java.


“DOWNLOAD”
“BROWSERS”

<http://www.netscape.com>

Raiz do hipertexto montado pelos criadores do Netscape Navigator™. Sob este endereço você pode obter o browser “Java compatible” da “Netscape Communications INC’.. Outros desenvolvedores estão lançando “browsers” compatíveis com Java.


“DOWNLOAD”
“BROWSERS”

<http://www.microsoft.com>

A microsoft™ licenciou a tecnologia Java e a incorporou em seu novo browser: Internet Explorer versão 3.0 ou superior.



BROWSERS: São uma categoria de programas que permitem você visualizar um documento criado em um certo padrão, no caso html (hipertext markup language). Atualmente os browsers tem se tornado complexos devido a quantidade de padrões existentes (ex. imagens .gif .jpg, etc). A linguagem Java pode contribuir para minimizar esta complexidade.

3. Características da linguagem

3.1. Parecida com C, C++

Java tem a aparência de C ou de C++, embora a filosofia da linguagem seja diferente. Por este motivo estaremos frequentemente fazendo comparações alguma destas linguagens. O leitor que programa em qualquer uma delas, ou em uma linguagem orientada a objetos, se sentirá mais a vontade e se tornará um bom programador Java em menos tempo.

Java também possui características herdadas de muitas outras linguagens de programação: Objective-C, Smalltalk, Eiffel, Modula-3, etc. Muitas das características desta linguagem não são totalmente novas. Java é uma feliz união de tecnologias testadas por vários centros de pesquisa e desenvolvimento de software.

3.2. Compilada

Um programa em Java é compilado para o chamado “byte-code”, que é próximo as instruções de máquina, mas não de uma máquina real. O “byte-code” é um código de uma máquina virtual idealizada pelos criadores da linguagem. Por isso Java pode ser mais rápida do que se fosse simplesmente interpretada.

3.3. Portável

Java foi criada para ser portável. O “byte-code” gerado pelo compilador para a sua aplicação específica pode ser transportado entre plataformas distintas que suportam Java (Solaris 2.3®, Windows-NT®, Windows-95®, Mac/Os etc). Não é necessário recompilar um programa para que ele rode numa máquina e sistema diferente, ao contrário do que acontece por exemplo com programas escritos em C e outras linguagens.

Esta portabilidade é importante para a criação de aplicações para a heterogênea internet. Muitos dos programas exemplo deste tutorial foram escritos e compilados numa plataforma Windows-95® e rodaram perfeitamente quando simplesmente copiados para uma plataforma Solaris 2.3®. Em Java um inteiro por exemplo, tem sempre 32 bits, independentemente da arquitetura. O próprio compilador Java é escrito em Java, de modo que ele é portável para qualquer sistema que possua o interpretador de “byte-codes”. Um exemplo de programa escrito em Java é o browser hotjava.

3.4. Orientada a Objetos

A portabilidade é uma das características que se inclui nos objetivos almejados por uma linguagem orientada a objetos. Em Java ela foi obtida de maneira inovadora com relação ao grupo atual de linguagens orientadas a objetos.

Java suporta herança, mas não herança múltipla. A ausência de herança múltipla pode ser compensada pelo uso de herança e interfaces, onde uma classe herda o comportamento de sua superclasse além de oferecer uma implementação para uma ou mais interfaces.

Java permite a criação de classes abstratas. Outra característica importante em linguagens orientadas a objetos é a segurança. Dada a sua importância o tópico foi escrito a parte.

3.5. Segura

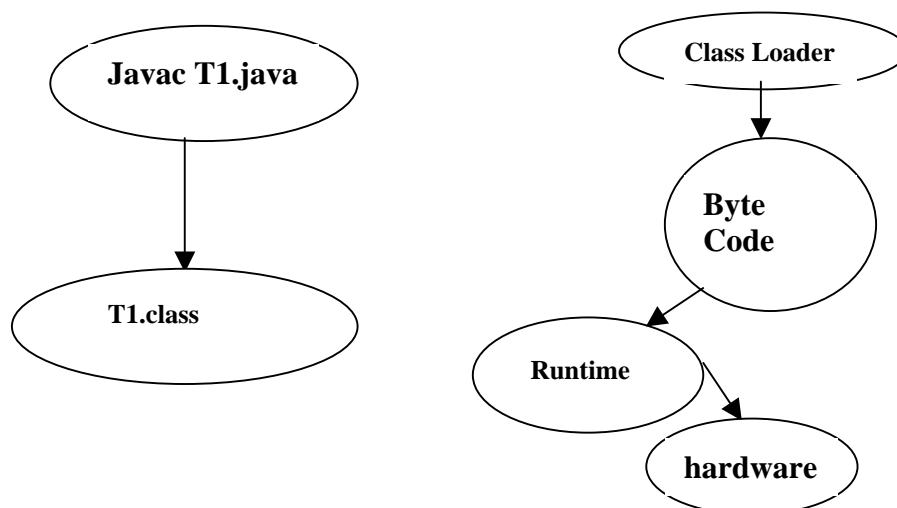
A presença de coleta automática de lixo, evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória (C, C++, Pascal). A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outras estruturas substitutivas traz benefícios em termos de segurança. O programador é proibido de obter acesso a memória que não pertence ao seu programa, além de não ter chances de cometer erros comuns tais como “reference aliasing” e uso indevido de aritmética de ponteiros. Estas medidas são particularmente úteis quando pensarmos em aplicações comerciais desenvolvidas para a internet.

Ser “strongly typed” também é uma vantagem em termos de segurança, que está aliada a eliminação de conversões implícitas de tipos de C++.

A presença de mecanismos de tratamento de exceções torna as aplicações mais robustas, não permitindo que elas abortem, mesmo quando rodando sob condições anormais. O tratamento de exceções será útil na segunda parte para modelar situações tais como falhas de transmissão e formatos incompatíveis de arquivos.

Os arquivos do Java são compilados e são convertidos de arquivos texto para um formato que contém blocos independentes de bytes codes (Código Intermediário).

Em tempo de execução estes bytes codes são carregados, são verificados através do Byte Code Verifier (uma espécie de segurança), passam a seguir para o interpretador e são executados. Caso este código seja acionado diversas vezes, existe um passo chamado JIT Code Generator, que elimina o utilização por demasia do tráfego da rede.



Abra o Notepad e crie o seguinte programa. Salve-o como **Prog0101.java**

```
class Prog0101
{
    public static void main (String arg [])
    {
        int a = 5, b = 10;
        a = a + 5;
        System.out.println("Meu Primeiro Progama");
        System.out.println(" O valor da variavel a = " + a);
    }
}
```

Após terminar o programa compile-o e execute-o:

```
C:\.....\Javac Progr0101.java
```

```
C:\.....\ Java Prog0101
```

3.6. Suporta concorrência

A linguagem permite a criação de maneira fácil, de vários “threads” de execução. Este tópico será útil quando você estudar animações, e é particularmente poderoso nos ambientes em que aplicações Java são suportadas, ambientes estes que geralmente podem mapear os threads da linguagem em processamento paralelo real.

3.7. Eficiente

Como Java foi criada para ser usada em computadores pequenos, ela exige pouco espaço, pouca memória. Java é muito mais eficiente que grande parte das linguagens de “scripting” existentes, embora seja cerca de 20 vezes mais lenta que C, o que não é um marco definitivo. Com a evolução da linguagem, serão criados geradores de “byte-codes” cada vez mais otimizados que trarão as marcas de performance da linguagem mais próximas das de C++ e C. Além disso um dia Java permitirá a possibilidade de gerar código executável de uma particular arquitetura “on the fly”, tudo a partir do “byte-code”.

3.8. Suporte para programação de sistemas distribuídos

Java fornece facilidades para programação com sockets, remote method call, tcp-ip, etc. Estes tópicos não serão abordados neste texto.

4. Tipos de Dados

4.1. Comentários

Estes são os três tipos permitidos de comentários nos programas feitos em Java:

```
// comentário de uma linha
/* comentário de uma ou mais linhas */
/** comentário de documentação */ (Arquivos de documentação)
```

4.2. Ponto e vírgula, Blocos e o espaço em branco

- No java, os comandos são terminados com o sinal de ponto e vírgula (;)
- Um bloco tem início e tem o seu fim representados pelo uso das chaves {};
- O uso do espaço em branco permite uma melhor visualização dos comandos e em consequência facilita a sua manutenção.

4.3. Identificadores

Na linguagem Java um identificador é startado com uma letra, underscore (_), ou sinal de dólar (\$), e existe uma diferenciação entre letras maiúsculas e minúsculas:

Identificadores válidos:

- identifier
- userName
- User_name
- _sys_var1
- \$change

4.4. Tipos Básicos no Java

No Java existem oito tipos básicos e um tipo especial.

4.4.1. Tipo Lógico

- boolean: on e off; true e false ou yes e no.

4.4.2. Tipo Textual

- char e String

Um caracter simples usa a representação do tipo char. O tipo char representa na forma Unicode um caracter de 16-bit.

O literal do tipo char pode ser representado com o uso do (' ').

`'\n'` – nova linha
`'\r'` – enter
`'\u????'` – especifica um caracter Unicode o qual é representado na forma Hexadecimal.

`'\t'` – tabulação
`'\|'` - \|
`'\"'` - “”

O tipo String, como não é primitivo, é usado para representar uma seqüência de caracteres.

4.5. Tipo Ponto Flutuante

Uma variável do tipo ponto flutuante pode ser declarada usando a palavra *float* ou *double*.

3.14	Um ponto flutuante simples;
6.02E23	Um valor de ponto flutuante largo;
2.718F	Um valor de ponto flutuante simples;
123.4E+306D	Um valor de ponto flutuante usando o tipo double.

5. Tipos de dados avançados

5.1. Declaração de Arrays

```
char s [ ];  
Point p [ ];
```

Em Java um Array é uma classe.

5.1.1. Criando um Array

Você pode criar arrays, ligando-o a todos os objetos, usando a palavra *new*, da seguinte forma:

```
s = new char[20];  
p = new Point[100];  
  
String names[ ];  
names = new String[4];  
names[0]= “Georgina”;  
names[1]=“Jen”;  
names[2]=“Simon”;  
names[3]= “Tom”;
```

ou

```
String names[ ];  
names = new String[4];  
String names [ ] = { "Georgina", "Jean", "Simon", "Tom"};
```

5.1.2. Arrays Multi-dimencionais

Java não possui arrays multi-dimencionais, mas ele permite declarar um array que é baseado em um outro array.

```
int twoDim [ ] [ ] = new int [4] [ ] ;  
twoDim[0] = new int [5] ;  
twoDim[1] = new int [5] ;
```

6. Variáveis e Tempo de vida

Você tem dois meios para descrever variáveis: usando o tipo simples de ligação *int* e *float* ou usando tipos de classes definidas pelo programa. Você pode declarar variáveis de duas formas, uma dentro de um método e a outra dentro da classe a qual este método está incluído.

6.1. Inicialização de variáveis

No Java não é permitido o uso de variáveis indefinidas.

Variáveis definidas dentro do método são chamadas de variáveis automáticas, locais, temporárias ou estáticas e devem ser inicializadas antes do uso.

Quando um objeto é criado, as variáveis membro são inicializadas com os seguintes valores em tempo de alocação:

Tipo de variável	Valor inicial	Tamanho
byte	0	8 bits
short	0	16 bits
Int	0	32 bits
long	0L	64 bits
float	0.0f	32 bits
Double	0.0d	64 bits
Char	'\u0000' (Null)	64 bits
Boolean	false	

7. Operadores

No Java os operadores são muito similares ao estilo e funcionalidade de outras linguagens como por exemplo o C e o C++.

Pré-incremento:

```
x = 10;  
    x = x + 1;  
    O valor da variável x é 11
```

ou

```
x = 10;  
++x  
O valor da variável x é 11.
```

Pós-Incremento:

```
x = 10;  
    x = x + 1;  
    O valor da variável x é 11
```

ou

```
x = 10;  
x++  
O valor da variável x é 11.
```

Diferença entre o Pré-Incremento e o Pós-Incremento:

```
x = 10  
++x => neste exato momento a variável a vale 11
```

```
x = 10  
x++ => neste exato momento a variável x vale 10
```

Separadores:

. [] () ; ,

Operadores:

Operadores	Descrição
==	Igualdade
!=	Negação
+ - * /	Aritméticos
&&	e
	ou

7.1. Concatenação

O operador + é utilizado para concatenar objetos do tipo String, produzindo uma nova String:

```
String PrimeiroNome = "Antonio";
String SegundoNome = "Carlos";
String Nome = PrimeiroNome + SegundoNome;
```

7.2. Casting (Conversão de tipos)

A linguagem Java não suporta conversões arbitrárias de tipos de variáveis. Você deve explicitar a conversão entre tipos de variáveis.

Exemplo:

```
long bigval = 6;           // Operação válida
int smallval = 99L;       // Operação inválida porque são de tipos diferentes

float z = 12.414F;        // Operação válida
float zp = 12.414;        // Inválido, porque está tentando atribuir um valor double.
```

Convertendo

```
Long bigValue = 99L;
Int squashed = (int)(bigValue);
```

8. Instruções de controle:

Declarações de Fluxos

If, else

```
if (expressão)           // expressão cujo retorno é um valor do tipo boolean
    { Declarações ou blocos }
else                       // caso a condição anterior não seja satisfeita
    { Declarações ou blocos }
```

switch

```
switch (expressão)      // Esta expressão deve ser do tipo int ou char
{
    case cond01:
        declarações;
        break;           // usado para sair do case.
```

```

        case cond02:
            declarações;
            break;
        case cond03:
            declarações;
            break;
    }

```

for Loops

```

for (expr_inicial; condição_boolean; incremento)
{
    Declarações ou blocos;
}

```

while Loops

```

while(condição_boolean)
{
    Declarações ou blocos;
}

```

do Loops

```

do
{
    Declarações ou blocos;
}
while(condição_boolean);

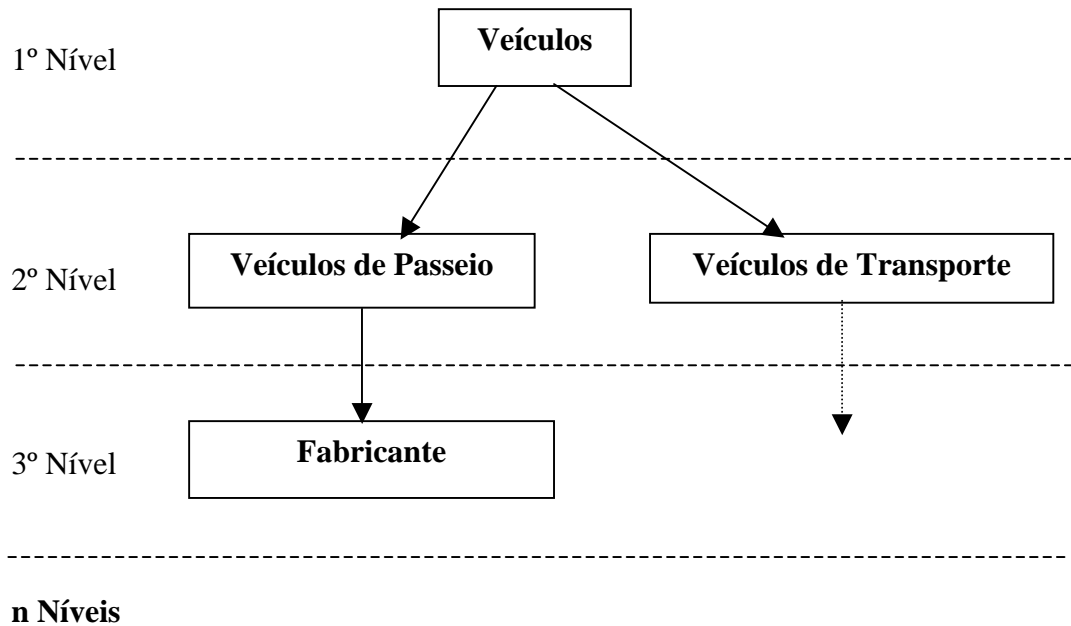
```

9. Palavras Reservadas

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

10. Estruturação teórica da linguagem:

10.1. Classes e objetos:



10.2. Abstração de Tipos de Dados

Quando itens de dados são compostos para tipos de dados, semelhante a uma data, você pode definir um número de bits de programas que especifica a operação do tipo de dados.

Em Java você pode criar uma associação entre o tipo data e a operação tomorrow a seguir:

```
public class Date {
    private int day, month, year;
    public void tomorrow( )
    { // código que incrementa o dia
    }
}
```

10.3. Definição de Métodos

Em Java, métodos são definidos usando uma aproximação que é muito similar à usada em outras linguagens, como por exemplo C e C++. A declaração é feita da seguinte forma:

< modifiers > <tipo de retorno> < nome > (< lista de argumentos >) < bloco >

< modifiers > -> segmento que possui os diferentes tipos de modificações incluindo *public*, *protected* e *private*.

< tipo de retorno > -> indica o tipo de retorno do método.

< nome > -> nome que identifica o método.

< lista de argumentos > -> todos os valores que serão passados como argumentos.

```
public void addDays (int days)
```

10.4. Passagem de Valores

Em Java o único argumento passado é “by-value”; este é um argumento *may not be changed* do método chamado. Quando um objeto é criado, é passado um argumento para o método e o valor deste argumento é uma referência do objeto. O conteúdo do objeto passível de alteração é chamado do método, mas o objeto referenciado jamais é alterado.

10.5. A Referência This

É aplicado a métodos não estáticos.

O Java associa automaticamente a todas as variáveis e métodos referenciados com a palavra *this*. Por isso, na maioria dos casos torna-se redundante o uso em todas as variáveis da palavra *this*.

Existem casos em se faz necessário o uso da palavra *this*. Por exemplo, você pode necessitar chamar apenas uma parte do método passando uma instância do argumento do objeto. (Chamar um classe de forma localizada);

```
    Birthday bDay = new Birthday(this);
```

10.6. Ocultando Dados

Usando a palavra *private* na declaração de *day*, *month* e *year* na classe *Date*, você impossibilitará o acesso a estes membros de um código fora desta classe. Você não terá permissão para atribuir valores, mas poderá comparar valores.

10.7. Encapsulamento

É uma proteção adicional dos dados do objeto de possíveis modificações impróprias, forçando o acesso a um nível mais baixo para tratamento do dados.

10.8. Sobrescrevendo Métodos

O Java permite que você tenha métodos com o mesmo nome, mas com assinaturas diferentes. Isto permite a reusabilidade dos nomes dos métodos.

```
public void print( int i )
public void print( float f )
public void print( String s)
```

Quando você escreve um código para chamar um desses métodos, o método a ser chamado será o que coincidir com tipos de dados da lista de parâmetros.

10.9. Construtores

O mecanismo de inicialização do Java é automático, ou seja se não inicializarmos um construtor, o Java o inicializará automaticamente.

Mas existem casos que se faz necessário a declaração explícita dos construtores.

Para escrever um método que chama um construtor, você deve seguir duas regras:

- 1ª O nome do método precisa ser igual ao nome da classe.
- 2ª Não deve retornar um tipo declarado para o método.

10.10. Subclasses (Relacionadas)

Na classe Pai você deve declarar os objetos comun a todos, e nos sub-níveis (Subclasses), você declara as particularidades:

```
public class Employee {
    private String name;
    private Date hireDate;
    private Date dateOfBirth;
    private String jobTitle;
    private int grade;
    .....
}
```

10.11. Subclasses (Extensões)

Em linguagens de orientação a objetos, um mecanismo especial é fornecido para que permita ao programa defina classes e termos previstas na definição de outras classes. Esta arquitetura em Java usa a palavra `extends`.

```

    public class Employee {
        private String name;
        private Date hireDate;
        private Date dateOfBirth;
        private String jobTitle;
        private int grade;
    }

    public class Manager extends Employee {
        private String departament;
        private Employee [ ] subordinates;
        .....
    }

```

10.12. Herança Simples

Em Java não existe herança múltipla.

Em Java os Construtores não são herdados.

Java permite que uma classe estenda uma outra classe, com isso esta classe herda as características da outra classe.

10.13. Polimorfismo

A idéia de polimorfismo é a de muitas formas, onde eu posso utilizar uma classe de diversas maneiras e formas possíveis.

```
public class Employee extends Object
```

and

```
public class Manager extends Employee
```

10.14. Argumentos de Métodos e Coleções Heterogêneas

Usando esta aproximação você pode escrever métodos que aceitam um objeto genérico. O uso do polimorfismo fornece uma série de facilidades.

```
public TaxRate findTaxRate( Employee e) {
    Manager m = new Manager( );
    .....
}

```

```
TaxRate t = findTaxRate(m);
```

Isto é possível porque um Gerente é um empregado.

Uma coleção heterogênea é uma coleção de coisas diferentes. Em linguagem orientada a objetos, você pode criar uma coleção de coisas que tem uma classe de antepassados comuns. Assim nós poderemos fazer isso.

```
Employee [ ] staff = new Employeee [ 1024 ];  
staff[ 0 ] = new Manager ( );  
staff[ 1 ] = new Employee ( );
```

E assim sucessivamente nos podemos escrever um método de tipos que põe empregados ordenados por idade ou em ordem salarial sem ter que se preocupar com a ordem de inserção.

10.15. O Operador instanceof

Fornece o dado que você adquiriu através da passagem de parâmetros.

Caso você receba um object por referência do tipo Employee, esta referência poderia não ser mostrado para Manager. Se você quiser testar isso use instanceof.

```
public void method(Employee e) {  
    if (e instanceof Manager) {  
  
    }  
    else if ( e instanceof Contractor) {  
  
    }  
    else {  
  
    }  
}
```

10.16. Objetos Lançados

Em circunstâncias onde você recebeu uma referência para uma classe pai, e você determinou que o objeto é de fato uma subdivisão de classe particular usando o operador de instanceof, você pode restabelecer a funcionalidade completa do objeto lançado.

10.17. Sobre-escrevendo Métodos

O Java permite que você declare métodos na classe pai e não desenvolva nenhuma lógica dentro desses métodos, permitindo com isso que o desenvolvimento desses métodos venha a ser trabalhados dentro das sub-classes posteriores.

Class Funcionario()

ler()

Class Motorista()

ler ()

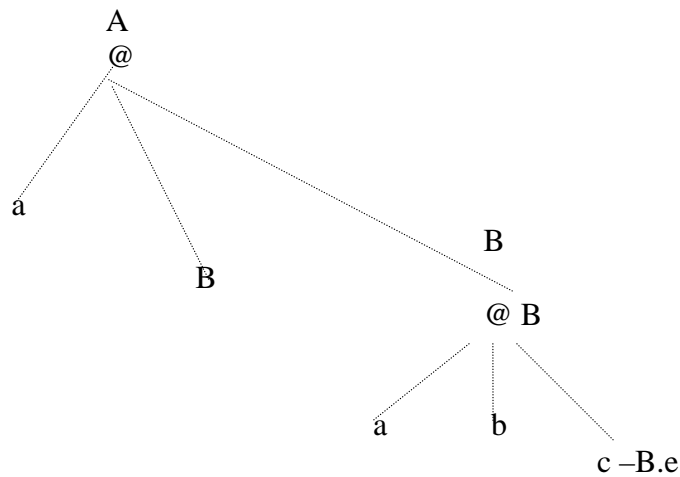
Super.ler() -> Referencia o método da classe pai

11. Comparando Java com outras Linguagens

Em C++ você pode adquirir este comportamento, desde que você marque o método como virtual na fonte.

11.1. Classes se agrupando - Pacotes

Java provê o mecanismo de pacotes como um modo de se agrupar classes relacionadas. Tão longe, todos nossos exemplos pertencem à falta ou pacotes não mencionados.



onde:

@ - Pacote
letra (a, b ...) - classe

11.2. A declaração import

Em Java, quando você quer usar instalações de pacotes, você usa a declaração de importação para contar para o compilador onde achar as classe que você vai usar.

A declaração de importação (import) deve preceder a declaração de todas as classes.

12. O que é um Applet?

Um applet é um pedaço de código de Java que corre em um ambiente de browser. Difere de uma aplicação do modo que é executado. Uma aplicação é começada quando seu método main() é chamado. Através de contraste, o ciclo de vida de um applet é um pouco mais complexo. Este módulo examina como um applet é corrido, como carregar isto no browser, e como escrever o código para um applet.

12.1. Carregando um Applet

Você tem que criar um arquivo de HTML que conta para o browser o que carregar e como correr isto. Você então "aponta" o browser ao URL que especifica aquele arquivo de HTML.

12.2. Segurança e Restrições nos Applet

Existem Seguranças e Restrições nos Applets porque applets são pedaços de código que são carregados em cima de arames, eles representam um prospecto perigoso; imagine se alguém escrever um programa malicioso que lê seu arquivo de contra-senha e envia isto em cima da Internet?

O modo que Java previne isto está contido na classe de SecurityManager que controla acesso para quase toda chamada de sistema-nível no Java Máquina Virtual (JDK).

Portanto Applets não faz chamadas ao seu sistema operacional.

12.3. Métodos de Applet chaves

Em uma aplicação, no programa é entrado ao método main(). Em um applet, porém, este não é o caso. O primeiro código que um applet executa é o código definido para sua inicialização, e seu construtor.

Depois que o construtor é completado, o browser chama um método no applet chamado init (). O método init () executar inicialização básica do applet . Depois de init () é completado, o browser chama outro método chamado start ().

12.4. Applet Display

Você pode puxar sobre a exibição de um applet criando um método `paint()`. O método `paint()` é chamado pelo ambiente de browser sempre que a exibição do applet precise ser refrescado. Por exemplo, isto acontece quando a janela de browser é aparecida depois de ser minimizado.

13. Ponteiros, “Pointers”, Referências e Objetos

Alguém pode ter achado estranho que não foram discutidos ponteiros em Java, ocorre que eles não existem nessa linguagem. Existem estudos que afirmam que erros com ponteiros são um dos principais geradores de “bugs” em programas, além disso com todos os recursos que temos não precisaremos deles.

Os programadores acostumados ao uso de ponteiros (e aos erros decorrentes desse uso), acharão muito natural e segura a transição para Java onde passarão a usar principalmente vetores e classes. A estruturação de seu código deverá agora ser feita em um modelo que se baseia no uso de objetos (vetores e Strings também são objetos). Objetos superam a representatividade obtida com records, funções isoladas e ponteiros.

De certo modo você estará usando referências, mas de forma implícita. Por exemplo: objetos são alocados dinamicamente com `new`, eles são referências ou ponteiros para posições na memória, mas a linguagem mascara este fato por razões de segurança. Como objetos são ponteiros (só que transparentes para você), nos depararemos com o problema de `reference aliasing` quando discutirmos cópia de objetos com outros objetos como atributos.

13.1. Passagem por referência

Linguagens como Pascal ou C criam meios de passar parâmetros por valor ou por referência. Como Java não possui ponteiros, a passagem por referência deve ser feita através de objetos. Se o parâmetro já é um objeto, então a passagem dele é obrigatoriamente por referência.

No caso de tipos simples, podemos passá-los dentro de vetores que são objetos, ou então criar classes para embalar, empacotar estes tipos. Dada a necessidade destas classes, elas já foram definidas na linguagem. São classes definidas para conter cada tipo básicos e permitir certas conversões entre eles, falaremos destas classes conforme necessitarmos de seu uso. As vezes estas classes são chamadas de “wrappers”.

13.2. Vetores e matrizes

Vetores são objetos, eles possuem papel importante no estilo de programação desta linguagem que exclui ponteiros. Por serem objetos, vetores são obrigatoriamente alocados de maneira dinâmica. O exemplo a seguir aloca um vetor de inteiros com três posições, seguindo uma sintaxe semelhante a de alocação de objetos:

CÓDIGO

```
class VetorTest {
    public static void main (String args[]) {
        int vetor[]=new int[3];
        vetor[0]=0; //indexação
        semelhante a C , C++
        vetor[1]=10;
        vetor[2]=20;
        System.out.println(vetor[0]+" "+vetor[1]+" "+vetor[2]+" ");
    }
}
```



0 10 20

Resumo da sintaxe de vetores:

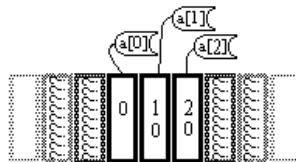
```
int a[]; //declara vetor de inteiros a
a=new int[10]; //aloca vetor a com dez posicoes
//as duas linhas anteriores podem ser abreviadas por:
int a[]=new int[10];
//alem disso se voce quiser inicializar o vetor a, ja' na declaracao:
int a[3]={0,10,20};
```

O análogo para matrizes é:

```
int a[][]; //declara matriz de inteiros a
a=new int[3][3]; //aloca matriz 3x3, 9 celulas
//as duas linhas anteriores podem ser abreviadas por:
int a[]=new int[3][3];
//alem disso se voce quiser inicializar a matriz a ja na declaracao:
int a[3][3]={{0,10,20},{30,40,50},{60,70,80}};
```

Em métodos, argumentos e valores de retorno que são vetores, são escritos da seguinte forma: `int[]`, ou `tipo[]` nomedavariavel //no caso de argumentos.

Diagrama do vetor:



Perceba que a faixa útil do vetor vai de 0 até (n-1) onde n é o valor dado como tamanho do vetor no momento de sua criação, no nosso caso 3. O mesmo ocorre com matrizes. Esta convenção pode confundir programadores Pascal onde a indexação vai de 1 até n.

14. Alocação Dinâmica

Este exemplo cria um tipo abstrato de dados matriz bidimensional de inteiros (int). O leitor pode achar estranho que para representar esta matriz usamos um vetor, mas é que isto traz algumas vantagens:

14.1. Representação linear de uma matriz

Pode-se representar uma matriz de qualquer dimensão em um vetor. Veja o exemplo de uma matriz bidimensional de inteiros mostrada no formato `indiceLinear:valorArmazenado`. Os índices lineares vão de 1 até n^2 .

Matriz:

1:3	2:32	3:1
4:23	5:90	6:12
7:21	8:08	9:32

Vetor equivalente:

1:3	2:32	3:1	4:23	5:90	6:12	7:21	8:08	9:32
-----	------	-----	------	------	------	------	------	------

Vantagem da representação linear (vetor): para referenciar uma posição gasta-se somente um inteiro contra dois da representação matriz. Pode-se considerar posições que apontam para outras posições, basta interpretar o conteúdo do vetor como um índice linear. Este tipo de construção pode ser útil em **Java**, pois a linguagem não possui ponteiros este é um dos motivos de estarmos ensinando esta técnica.

Desvantagem da representação linear (vetor): é necessário criar funções de conversão de índice na forma (linha,coluna) para (índice linear) e de (índice linear) para (coluna) ou (linha). São as funções `lin` e `col` e `linear` deste exemplo. Em uma primeira leitura, não é preciso entender os cálculos com índices, apenas o uso dos métodos que oferecem estes cálculos.

Para nós, clientes da classe `Matriz2DInt`, os elementos serão indexados de 1 até m (arbitrário) em termos de índice linear. Em termos de linhas e colunas, eles serão indexados de (1,lmax) e de (1,cmax). O fato da linguagem adotar índices de 0 até $m-1$ para matrizes e vetores não importa, nós construímos em volta dessa representação para que nosso objeto forneça, trabalhe na convenção mais natural de indexação para humanos: 1 até m . Quanto as operações de índices, apenas verifique a veracidade para valores arbitrários de uma matriz como a desenhada anteriormente tentar entendê-las leva tempo e eu sei que você é capaz de programa-las.

Dentre os objetivos de um programador de uma linguagem orientada a objetos podemos citar: escrever pouco código, escrever código correto, tornar o seu código reutilizável. A criação de componentes de software reutilizáveis, é enormemente facilitada pela portabilidade da linguagem **Java**. Programas exemplo posteriores (segunda parte) mostrarão como reutilizar esta classe matriz para a criação de um jogo de quebra cabeça de quadradinhos deslizantes. Os quadradinhos devem ser movidos na moldura de modo a formar uma imagem onde um dos quadrados é vazio.

14.2. Matrizes definidas na linguagem:

As matrizes definidas pela linguagem seguem uma sintaxe de declaração e uso semelhante a sintaxe de vetores:

```
int custos[][]=new int [20,30]; //vinte por trinta, não importa qual e linha qual e coluna
custos[0][0]=145;
int a=custos[0][0];
```

15. Conclusão (Visão geral da linguagem):

O Java baseou-se no C++, mas foi especificamente projetado para ser menor mais simples e mais confiável. O Java tem tantos tipos como classes. Tipos primitivos não são objetos baseados em classes. Esses incluem todos os seus tipos escalares, inclusive para dados inteiros, de ponto-flutuante, booleanos e de caracteres. Os objetos são acessados por variáveis de referências, mas os valores de tipos primitivos são acessados exatamente como os valores escalares em linguagens puramente imperativas como o C e a Ada. Os arrays Java são instâncias de uma classe predefinida, enquanto que no C++ eles não são.

O Java não tem ponteiros, mas seus tipos de referências oferecem alguns tipos de capacidades dos ponteiros. Essas referências são usadas para apontar instâncias de classes, sendo esta a única maneira delas serem referenciadas. Embora os ponteiros e as referências possam ser bastante semelhantes, existem algumas importantes diferenças semânticas. Aqueles indicam localização da memória, mas essas apontam para objetos. Isso torna sem sentido qualquer tipo de operação aritmética sobre as referências, eliminando essa prática propensa a erros. A distinção entre o valor de um ponteiro e o valor para onde ele aponta cabe ao programador determinar. As referências são implicitamente “desreferenciadas”, quando necessário. Assim elas se comportam como variáveis escalares comuns.

O Java tem um tipo booleano primitivo, usado principalmente para expressões de controle de suas instruções de controle, por exemplo **if** e **while**. Diferentemente do C e do C++, expressões aritméticas podem ser usadas para expressões de controle. O Java não tem nenhum tipo de registro, união ou enumeração.

Uma diferença entre o Java e muitas de suas contemporâneas que suportam a programação orientada a objeto é que não é possível escrever subprogramas independentes em Java. Todos os subprogramas Java são métodos e definidos em classes. Não há nenhuma construção de chamada de função ou de um subprograma. Além disso, métodos somente podem ser chamados por intermédio de uma classe ou objeto.

Outra importante diferença entre o C++ e o Java é que o C++ suporta herança múltipla diretamente em suas definições de classe. Alguns acham que herança múltipla acarretam mais complexidade e confusão do que deve. O Java suporta somente herança única, embora alguns benefícios daquela poderem ser ganhos, usando-se sua construção de interface.

O Java inclui uma forma relativamente simples de controle de concorrência por meio de seu modificador **synchronize**, que pode aparecer em métodos ou em blocos. Em qualquer um dos casos ele faz que um bloqueio seja anexado. O bloqueio garante o acesso ou execução mutuamente exclusivos. Em Java, é relativamente fácil de criar

processos concorrentes chamados threads que podem ser iniciados, suspensos, retomados e interrompidos , tudo com os métodos herdados da classe principal deles.

O Java usa desalocação de armazenagem implícita para seus objetos alocados da pilha, muitas vezes chamada de coleta de lixo. Isso libera o programador de preocupar-se em devolver o espaço de armazenagem à pilha quando ele não for mais necessário. Programas escritos em linguagens que exigem desalocação implícita freqüentemente padecem daquilo que, as vezes, é chamado de vazamento de memória, o que significa que um espaço é alocado, mas nunca é desalocado. Isso pode levar evidentemente ao total esgotamento de memória disponível. Diferentemente do C e do C++, o Java inclui coerções de tipo somente se eles tiverem ampliando-se . Assim coerções (conversões do tipo implícitas) somente se eles tiverem ampliando-se (de um tipo menor para um maior). Assim , coerções de **int** para **float** são feitas mas coerções de **float** para **int** não.