JAVA E ORIENTAÇÃO A OBJETOS (O.O)

BREVE HISTÓRIA

Java é uma linguagem de programação orientada a objetos, conceitualizada em 1991 e lançada oficialmente em 1995 pela Sun Microsystems. (ORACLE)

“A Sun Microsystems, em 1991 financiou um projeto de pesquisa corporativa interna com o codnome green, que resultou no desenvolvimento de uma linguagem baseada em C++ que seu criador, James Gosling chamou de OAK.” (DEITEL, PG.6)

Deitel esclarece ainda, que mais tarde descobriu-se que já existia uma linguagem com este nome e em visita a uma cafeteria local a equipe da sun escolheu o nome java em homengaem a uma cidade de origem de um tipo de café importado servido no estabelecimento. O java evoluiu a partir do C++ (linguagem estruturada e orientada a objetos) que evoluiui do C (linguagem estruturada)

LINGUAGEM ESTRUTURADA VS ORIENTADA A OBJETOS

PARADIGMA DA LINGUAGEM ESTRUTURADA

“É um tipo de programação orientada para a ação, sendo que a unidade de programação é a função. grupos de ações que realizam alguma tarefa comum são reunidos em funções e as funções são agrupadas para formar programas.” (DEITEL, PG.15)

PARADIGMA DA LINGUAGEM ORIENTADA A OBJETOS

A idéia da orientação a objetos é desenvolver programas baseados em objetos do mundo real. Para onde você olhar irá se deparar com objetos, sejam eles animados ou inanimados: pessoas, animais, carros, motos, etc. Os objetos possuem: caracteristicas - tamanho, forma, cor, etc comportamentos – andar, falar, comer, etc. Na linguagem O.O a unidade de programação é a CLASSE a partir do qual os objetos são instanciados (criados)

VANTAGENS DA LINGUAGEM JAVA

- Open Source;
- Orientada a Objetos;
- Simples: modelada originalmente a partir da linguagem C++
- Segura: verificação em tempo de execução e de compilação; identificação rápida de erros. Recursos de segurança de acesso a arquivos e à rede;
- Desempenho: coletor automático de lixo para liberar a memória não usada (Garbage Collector); Interpretação em tempo de execução (JIT)

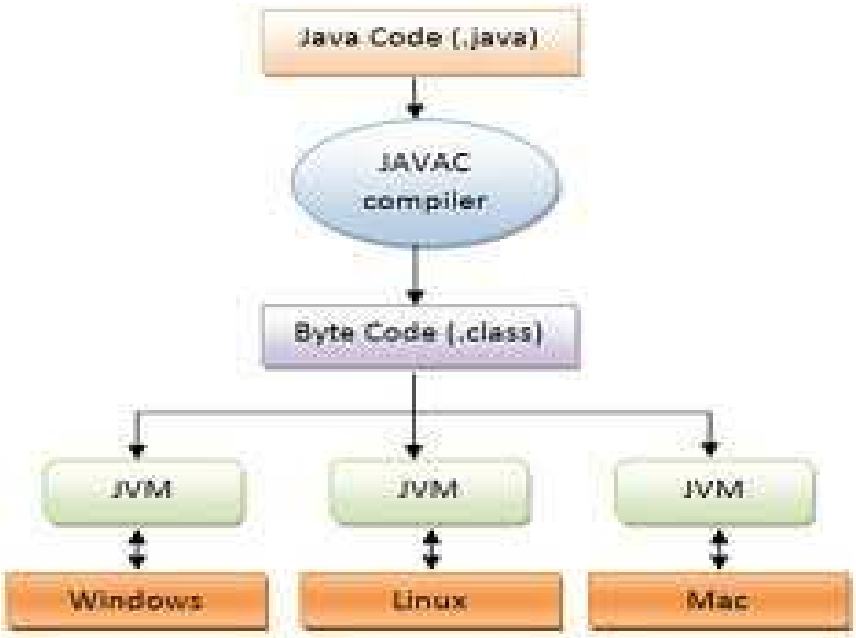
- Portátil: **“Write once run anywhere”**

Presente em mais de 7 bilhões de dispositivos, incluindo:

- Mais de 1 bilhão de computadores pessoais
- Cerca de 3 bilhões de telefones celulares
- Cerca de 3,5 bilhões de cartões inteligentes
- além de jogos, sistemas de navegação, etc

  Mais informações disponíveis em:
  http://www.java.com/pt_BR/download/faq/whatis_java.xml

COMPILAÇÃO



Fonte: http://viralpatel.net/blogs/2008/12/java-virtual-machine-an-inside-story.htm

JVM (Java Virtual Machine)

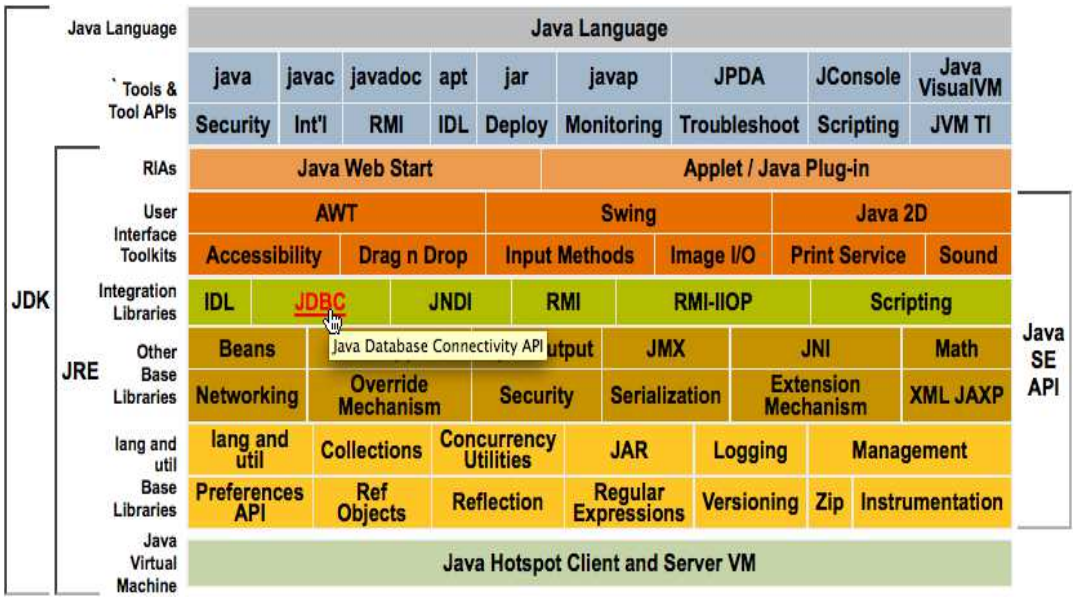
Máquina virtual Java, é um interpretador que confere portabilidade ao Java. Não existe Download da JVM

JRE (Java Runtime Environment)

É o ambiente para execução de aplicações Java. O download deve ser feito em função do Sistema Operacional. Este pacote contém a JVM e outras bibliotecas necessárias a execução de aplicações Java.

JDK (Java Development Kit)

Kit de desenvolvimento. Este pacote contém o pacote JRE além do compilador Javac, entre outros.



Fonte: http://netbeans.org/kb/docs/javaee/ecommerce/intro_pt_BR.html

DECLARAÇÃO DE VARIÁVEIS

Java é uma linguagem fortemente tipada, isto significa que diferentemente de outras linguagens, ao se declarar uma variável devemos dizer de que tipo ela é (Tipagem Estática). As operações são voltadas para estruturas de dados bem definidas. Em linguagens fracamente tipadas, as operações são aplicadas em qualquer estrutura de dados (Tipagem Dinâmica)

Exemplo de linguagens fracamente tipadas: Python, Ruby e PHP

Exemplo:

JAVA

```
public double soma(double x, double y) {
        return x+y;
}
```

RUBY (fracamente tipada)

```
def soma (x, y)
        return x + y
end
```

Os tipos de variáveis podem ser PRIMITIVOS ou de REFERÊNCIA

TIPOS PRIMITIVOS

```
byte bit = 127;
short numeroCurto = 32767;
int numero = -2.147.483.648;
long numeroLongo = 9147483648123111987L;
float valor = 1.67f;
double medida = 10.5;
char letra = 'a';
boolean situacao = true;
```

byte -- Inteiro de 8 bits. Pode assumir valores entre: -128 e 127.
short -- Inteiro de 16 bits. Pode assumir valores entre: -32.768 a 32.767
int -- Inteiro de 32 bits. Pode assumir valores entre: -2.147.483.648 e 2.147.483.647
long -- Inteiro de 64 bits. Pode assumir valores entre -2^{63} e 2^{63} -1.
float -- Ponto flutuante em precisão simples de 32 bits (IEEE 754-1985).
double -- Ponto flutuante em precisão dupla de 64 bits (IEEE 754-1985).
char – Caractere de notação UNICODE 16 bits
boolean -- 1 bit. Pode assumir valor true ou false;

TIPOS DE REFERÊNCIA

Característica específica de Linguagem O.O, em função da existência de CLASSES.
Ao criarmos uma classe podemos nos utilizar da mesma para a criação de objetos e para a declaração de variáveis do tipo da classe, como veremos mais para frente.

CASTING

Casting permite a conversão de alguns tipos primitivos e de referência.

Exemplos:

```
numero = (int) numeroLongo;
numeroCurto = (short) numeroLongo;
```

OPERAÇÕES

Principais Operadores utilizados em Java

= (atribuição de valores para variáveis)

== (igualdade de valores entre variáveis)

!= (diferença de valores entre variáveis)

! (negação)

&& (e)

|| (ou)

+ + (o mesmo que: variável = variable + 1)

- - (o mesmo que: variável = variável - 1

Além destes operadores, são aceitos os operadores matemáticos de soma, subtração, divisão, multiplicação, maior (>) e menor (<) e % para resto da divisão.

CLASSES

A classe é o projeto (receita) para a criação de objetos, contém atributos (variáveis) que são as característicasm, os métodos (semelhantes a funções) que são os comportamentos do objeto a ser criado e um construtor, que exige passagem de certos parâmetros para a criação do objeto em questão. As classes estão para os objetos assim como a planta está para a casa. Pode-se construir muitas casas a partir de uma planta, entretanto você não mora na planta e sim na casa, eis a diferença.

Em um outro exemplo, podemos fazer a analogia onde a classe seria a receita de uma torta e o objeto a torta, pode-se fazer muitas tortas com uma única receita. Você não come a receita e sim a torta.

Classes NÃO são objetos.

OBS: Por convenção de códigos, classes devem iniciar com letras Maiúsculas

CRIAÇÃO DE MÉTODOS

Os métodos devem possuir modificadores de acesso (veremos mais adiante), um retorno que pode ser vazio (VOID) ou pode retornar um INT, STRING, até mesmo um OBJETO, entre outros, alem de um nome.

```
public void imprimir() {
System.out.println("Hello World");
}
public double somar (double valor1, double valor2) {
double soma = valor1 + valor2;
return soma;
```

```
}
```

O construtor pode ser vazio ( ), ou seja, não recebe nenhum argumento, caso contrário devem ser declaradas váriavéis locais que serão o parâmetro do método.

CRIAÇÃO DO CONSTRUTOR

Os objetos aos serem criados recebem valores default em seus atributos. Uma conta pode receber um valor 0.0 na sua variável saldo, por exemplo. No entanto, pode acontecer (dependendo de sua regra de negócio) que você deseje que o responsável por criar contas obrigatoriamente atribua valores para certas variáveis do Objeto. Ex: para criar uma conta, esta deve possuir um cliente.

O construtor exige a passagem de certos parâmetros para a criação do objeto em questão.

Exemplo de Classe com atributos, métodos e construtor

```
public class Carros {

        // atributos

        String cor;

        String modelo;

        double velocidade;

        int marcha;


//construtor

public Carros (String cores, int marchas){

        this.cor=cores;

        this.marcha=marchas;

        this.modelo= ”Ferrari”;

        this.velocidade=0.0;

}

                // métodos

                public void ligar (){

                System.out.println(“Carro ligado”);
```

```
        }

        public void acelerar(double valor) {

        this.velocidade+= valor;

        }

}
```

METODO MAIN

O Main é o método principal que deve estar contido em uma única classe do seu projeto. É a porta de entrada da aplicação. Nele faremos a instância de nossos objetos e demais relações.

```
public static void main (String[] args){

}
```

INSTâNCIA (CRIAÇÃO) OBJETOS

Criar um objeto é simples. Após criar a classe que será o projeto para criação de objetos, você deve criar uma outra classe contendo um método main. No corpo do main utilize a seguinte instrução:

```
Carros carro1 = new Carros(  );
```

Para utilizar os métodos basta chamar pela variável

```
Ex:   carro1.ligar(  );

      carro1.acelerar(50);
```

A palavra **NEW**, cria um novo objeto do tipo da classe que vocë desejar. A referëncia ao objeto (endereço) deve ser armazenada em uma variável que deve ser do tipo do Objeto (variável de referência)

ENCAPSULAMENTO

Uma das carcterísticas da orientação a objetos é permitir o encapsulamento (empacotamento) dos atributos de uma classe. Deste modo, só é possivel manipular os atributos através de regras bem definidas implementadas nos métodos.

Ex: uma classe que cria objetos contas bancárias, deverá ter um atributo (variável) saldo que somente poderá ser acessado pelos métodos, saca, deposita, transfere, etc. Neste caso a varivél saldo deve receber o atributo private, tornando impossível a manipulação direta na variavél.

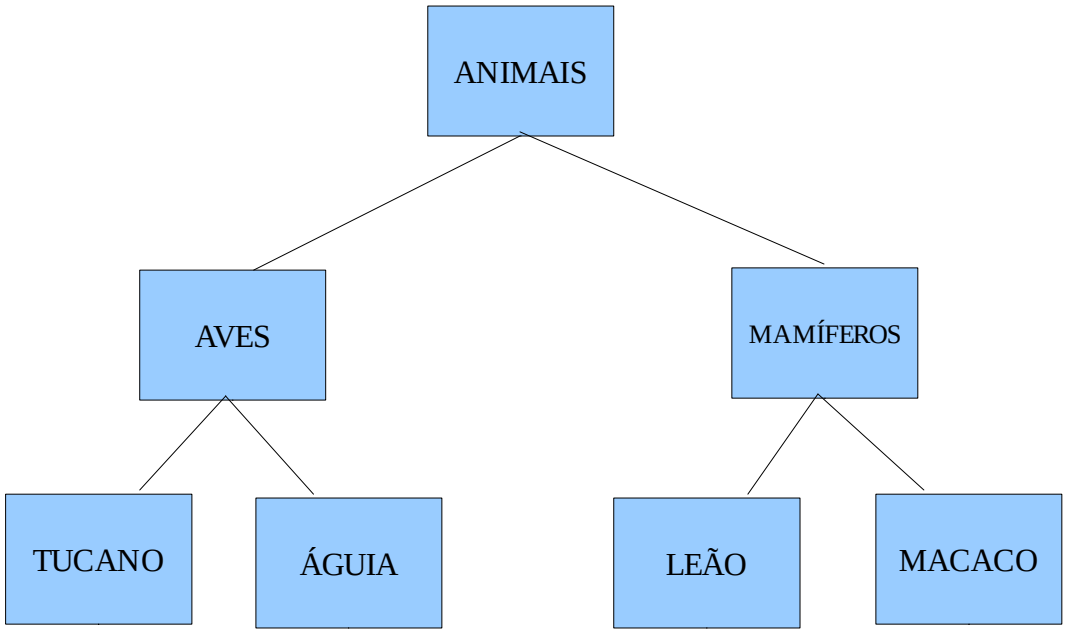
Ps: Somente os métodos daquela classe poderão acessar a variavél definida como private.

HERANÇA e POLIMORFISMO

A Herança e o Polimorfismo, são características de Linguagens Orientada a Objetos

Neste exemplo, temos a Classe mãe (ANIMAIS), com atributos e métodos característicos de todos os animais, como comer, por exemplo. As Classes AVES e MAMÍFEROS, herdam características e métodos da classe ANIMAIS, que por sua vez oferecerão recursos as demais classes.

Devemos utilizar a clausula **EXTENDS**, para que uma classe herde de outra.



Exemplo:

```
public abstract class Animais {
    //atributos e métodos
}
public abstract class Aves extends Animais{
    //herda atributos e métodos da classe Animais
    //atributos e métodos específicos da classe Aves
}
public class Tucano extends Aves {
```

```
        //herda atributos e métodos da classe Animais
        //herda atributos e métodos da classe Aves
        //atributos e métodos específicos da classe Tucano
    }
```

Para a utilização da herança, é importante a elaboração de um projeto que abranja a complexidade do sistema e suas possíveis extensões futuras. Toda classe mãe deve ser o mais genérica possível, afim de abranger as características de suas classes filhas.

A herança permite a utilização de um recurso denominado Polimorfismo.

O Polimorfismo permite a criação de métodos que recebem parâmetros genéricos, tornando-o mais flexível.

Exemplo:

```
public class Cuidador {
    public void alimentarAves (Aves aves){
    Se Tucano, alimentar com frutas
    Senão se Aguia, alimentar com camundongos
    }
}
```

Posso passar como parâmetro para o método qualquer tipo de ave, pois tucanos são aves e águias também.

Obs: A utilização da cláusula PRIVATE não permite a herança de atributos, neste caso devemos utilizar a clausula **PROTECTED** nas classes mães.

A cláusula **ABSTRACT** utilizada como modificador de acesso na classe, tornará impossível a criação de objetos Animais e Aves, o que faz sentido, pois nosso objetivo deve ser criar objetos ESPECÍFICOS e não GENÈRICOS.

Ps. quando cláusula ABSTRACT é utilizada no método, força a reescrita do método nas classes filhas.

INTERFACES

Em uma aplicação, as vezes faz sentido utilizar a herança, entretanto em algumas situações, determinadas classes necessitam herdar apenas alguns métodos específicos

da classe mãe, ou seja, não possuem correlação com a classe mãe para herdar suas características.

Imagine a situação na qual temos uma classe mãe Funcionários e as classes filhas Gerente e Tesoureiro, ambos sendo funcionários. A classe mãe possui um método específico de manipulação do sistema. Imagine uma outra classe denominada Cliente que precisa implementar o método manipulação do sistema. Como utilizar a herança se Cliente não é funcionário. Simples, neste caso invés de escrevermos o método na classe mãe Funcionário, escrevemos uma **INTERFACE** com este método e utilizamos a palavra **IMPLEMENTS** para Gerente, Tesoureiro e Cliente

Ex:

// Escrevemos a interface como se fosse uma nova classe, entretanto utilizamos interface ao invés de class

```
public interface Sistema {

// o método manipulação do sistema

}

// para implementar

public class Gerente extends Funcionarios implements Sistema {

// herda atributos e métodos de Funcionarios, e implementa o método manipulação do sistema

}

public class Tesoureiro extends Funcionarios implements Sistema {

// herda atributos e métodos de Funcionarios, e implementa o método manipulação do sistema

}

public class Cliente implements Sistema {

// implementa o método manipulação do sistema

}
```

OUTROS MODIFICADORES DE ACESSO

STATIC - este modificador de acesso, quando utilizado em uma variável torna o atributo como sendo da Classe e não do Objeto. Ex: ao criarmos uma classe responsável pela criação tortas, podemos criar um atributo STATIC que será um atributo da classe e não do

Objeto torta. Este atributo deve ser acessado pela classe e não pelo Objeto. Poderíamos utilizar este atributo para ser o contador da produção de tortas. Cada novo objeto criado, irá icrementar o contador.

```
public class Torta {
        private static int contador;
        // outros atributos e métodos
        // construtor
        public Torta (parämetros do construtor) {
        contador ++;
        }
        // método estático para acessar a variavel
        public static void contador( ) {
        System.out.println(contador);
        }
}
public class Executavel {
        public static void main(String[] args) {
        //criação de um objeto torta
        Torta torta1 = new Torta(parämetros do contrutor);

        // deve exibir 1 (criamos um objeto torta)
        Torta.contador(  );
        }
}
```

FINAL – O final normalmente é utilizado em variáveis para a criação de constantes. O valor atribuído a determinada variável com esta clausula não poderá ser alterado. Nos métodos, impede que eles sejam herdados. Na classe impede que ela seja herdada.

REFERÊNCIAS:

Furgeri, Sergio. Java 6: Ensino Didático: Desenvolvendo e Implementando Aplicações. 2 ed. - São Paulo: Érica, 2008.

Deitel, H.M. Java: como programar. 6 ed. - São Paulo: Pearson Prentice Hall, 2005

Sierra, Kathy; Bates, Bert. Use a Cabeça Java. 2 ed. - Rio de Janeiro: Ed. Alta Books Ltda, 2007.

Caelum Ensino e Inovação, disponível em: http://www.caelum.com.br/