UNIP – Universidade Paulista
Laboratório de Prática de Programação

ICET - Ciência da Computação

Prof. Fábio Luís Pereira

São Paulo
2008

Sumário

1. Introdução ao Java

1.1. *Java - Histórico*

Em 1992 a *Sun* criou uma equipe liderada por aquele que se tornou o pai do Java (James Gosling), com o objetivo de desenvolver inovações tecnológicas.

Desta forma, esta equipe propôs um interpretador para diversos dispositivos eletrônicos, tais como televisão e aparelho de TV a cabo, cujo fim era facilitar a reutilização do código. Porém, em princípio a idéia deste interpretador não obteve sucesso no mercado e foi somente com o surgimento da web que foi possível o lançamento do Java 1.0, afinal, considerando a diversidade de plataformas existentes, o Java possibilitava a programação uma única vez podendo o mesmo código ser executado em qualquer browser ou sistema operacional.

É importante destacar que o Java está focado em aplicações corporativas de médio à grande (nas quais o trabalho é realizado em equipe e não individualmente) e que, portanto, podem crescer. Neste sentido, há a necessidade de entender facilmente os códigos, havendo muita conectividade, bem como diversas plataformas.

Além disso, apesar do estigma de baixa produtividade, o Java fornece grande quantidade de bibliotecas gratuitas, facilitando o desenvolvimento de aplicações que utilizam desde tocadores de vídeo até relatórios, sistemas de buscas e impressão, dentre outros. Logo, o Java viabiliza a criação de aplicações sofisticadas sem a necessidade de adquirir caríssimos componentes específicos, isto é, reduz o custo do desenvolvimento por se tratar de uma linguagem gratuita.

1.2. *Java - Conceituação*

O Java apresenta como principais características Orientação a Objeto, Reutilização de Código e Portabilidade.

Aprofundando a reflexão acerca desta tecnologia, se faz necessário esclarecer que a mesma não se restringe somente a linguagem ou interpretador ou plataformas, mas diz respeito ao conjunto que engloba todos estes, isto é, que abarca linguagem, interpretador e plataformas.

Diante disto a tecnologia Java divide-se da seguinte forma:

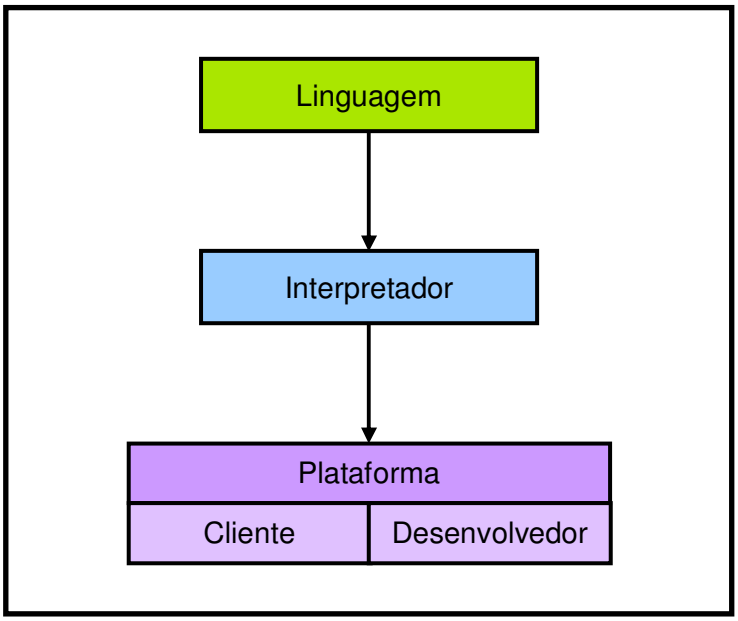


Figura 1: Tecnologia Java

1.2.1. A Saga das Versões

Grande parte da confusão dos nomes e versões do Java se dá, principalmente, por questões de *marketing* da Sun.

As versões 1.0 e 1.1 do Java não são utilizadas atualmente devido insuficiência de recursos frente às necessidades atuais.

Foi então que para diminuir a confusão entre Javascript e Java, além de várias melhorias e a incorporação de uma grande quantidade de funcionalidades a Sun resolveu trocar o nome Java 1.2 para Java 2, ficando assim Java 2 1.2. Mais tarde vieram as versões Java 2 1.3 e Java 2 1.4. Somando-se ao nome de cada uma destas versões um terceiro e/ou quarto número para indicar melhorias e correção de erros, conforme exemplo a seguir:

- Java 2 1.2.1
- Java 2 1.3.1_21
- Java 2 1.4.1, Java 2 1.4.2 ... Java 2 1.4.2_16

Na versão seguinte que seria Java 2 1.5, novamente por questões de *marketing* e mudanças técnicas significantes, se tornou Java 5. Atualmente o Java se encontra na versão 1.6, chamada de Java 6. Para indicar melhorias e correções de erros a Sun utiliza, a partir da versão do Java 5, a palavra Update e seu número seqüencial, conforme segue abaixo:

- Java 5, Java 5 Update 1, ... Java 5 Update 14
- Java 6, Java 6 Update 1, ... Java 6 Update 4

Uma das características mais benéficas do Java é a capacidade de compatibilidade, sendo assim, um código escrito em Java 2 1.2 deve necessariamente executar em Java 6 sem que ocorra alteração no código fonte.

1.2.2. As Plataformas Java

A plataforma Java é composta por três produtos, cada qual destinado a um propósito, segue abaixo uma breve descrição:

J2ME - Java 2 Micro Edition: plataforma direcionada para pequenos dispositivos como, telefones celulares, agendas eletrônicas, televisores, aparelhos eletrônicos em geral.

J2SE - Java 2 Standard Edition: plataforma voltada para aplicações cliente. Essa é a plataforma que será encontrada nos browsers web e instalada nos sistemas operacionais que já incluírem a plataforma Java.

J2EE - Java 2 Enterprise Edition: plataforma indicada para desenvolvimento e execução de aplicações servidoras. Possui suporte para desenvolver aplicações robustas e escaláveis, atende grandes números de usuários.

1.3. *Java - Jvm*

É aqui que vamos entender a inter-relação dos elementos desta tecnologia, que se baseia na seguinte topologia:

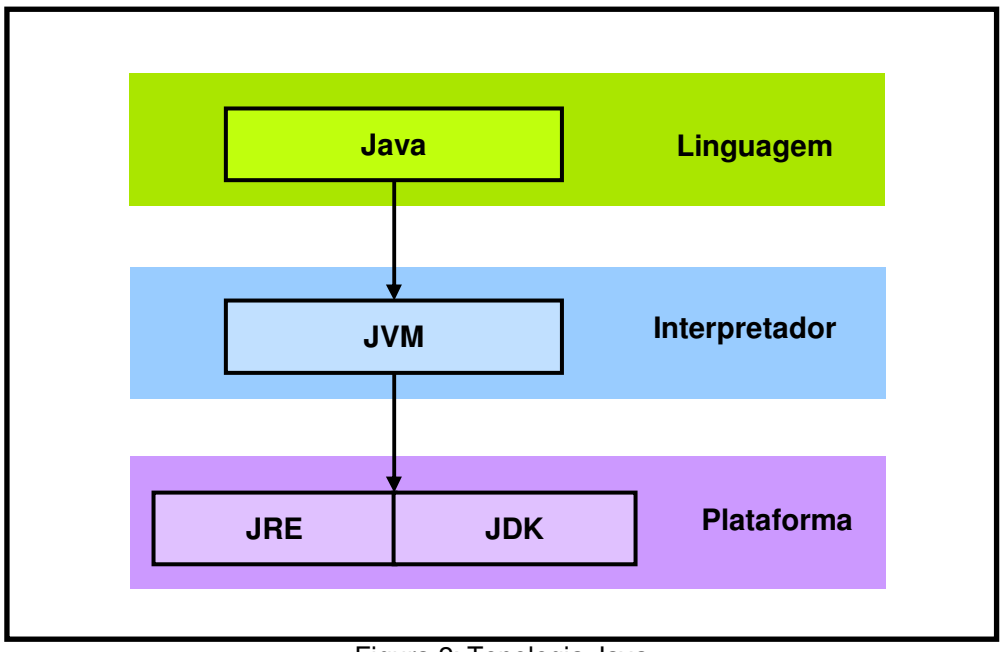


Figura 2: Topologia Java

Os elementos JDK, JRE, JVM e a linguagem Java, são explicados a seguir:

JDK (*Java Development Kit*[1]): é o conjunto básico de ferramentas para o desenvolvedor Java. Sendo tais ferramentas: javac, o compilador; Java, o executor; javadoc, o gerador automático de documentação; jdb, o depurador de erros[2]; javap, o decompilador; entre outras ferramentas importantes.

JRE (*Java Runtime Environment*[3]): é composto pela Java Virtual Machine, o conjunto de bibliotecas para execução de aplicações Java no cliente e também o Java Plugin, a JVM dos browsers web que permite browsers antigos suportem aplicações Java 2.

JVM (*Java Virtual Machine*[4]): é o interpretador, ou seja, uma espécie de tradutor do código para o sistema operacional é ainda responsável por carregar as classes do programa e verificar a integridade e a segurança do sistema.

A JVM é uma camada que se encontra entre a aplicação e o sistema operacional, onde a classe da aplicação é interpretada, sendo assim esta será traduzida para o sistema operacional sem que haja envolvimento direto com o

[1] Kit de Desenvolvimento Java
[2] Debugger
[3] Ambiente de execução Java
[4] Máquina Virtual Java

mesmo. Tornando sua execução segura e prevenindo interferências em outras aplicações caso ocorra algum erro.

Em outras linguagens, como o C e o Pascal, o código fonte é compilado para um sistema operacional específico. O arquivo binário[5] resultante da compilação destas linguagens muitas vezes utiliza as bibliotecas do próprio sistema operacional fazendo com que possa ser necessário reescrever o mesmo código para diferentes sistemas operacionais.

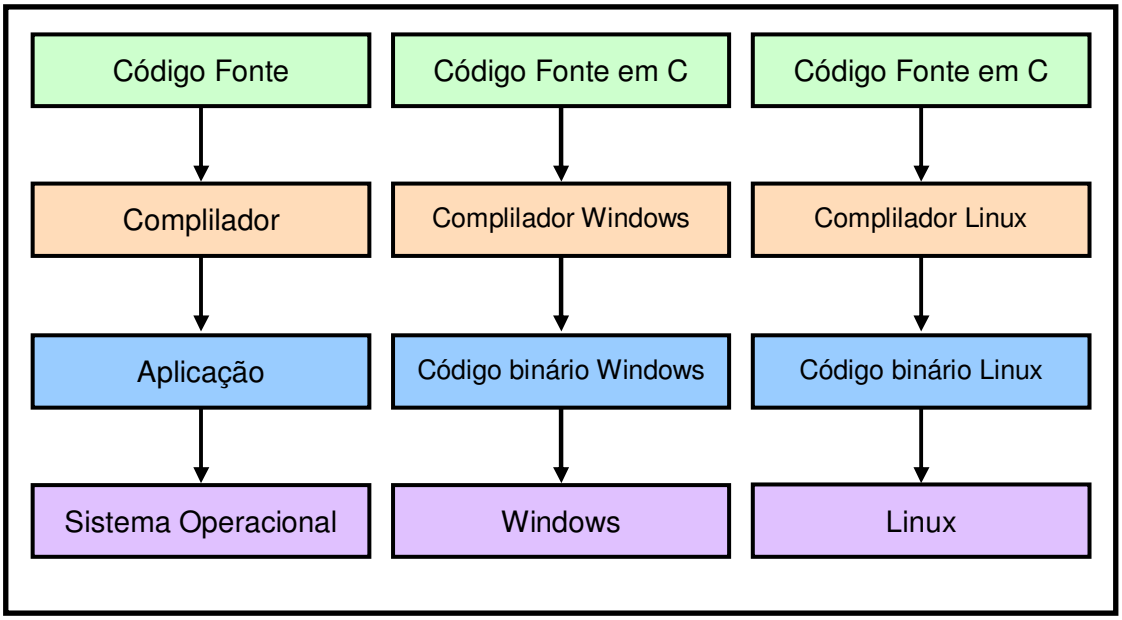


Figura 3: Arquitetura da Linguagem C

Com a utilização da JVM a independência do sistema operacional possibilita fazer medições e decidir o melhor abocamento de memória entre outros benefícios, além de verificar critérios e restrições de segurança.

[5] Conjunto de instruções que o processador é capaz de executar.

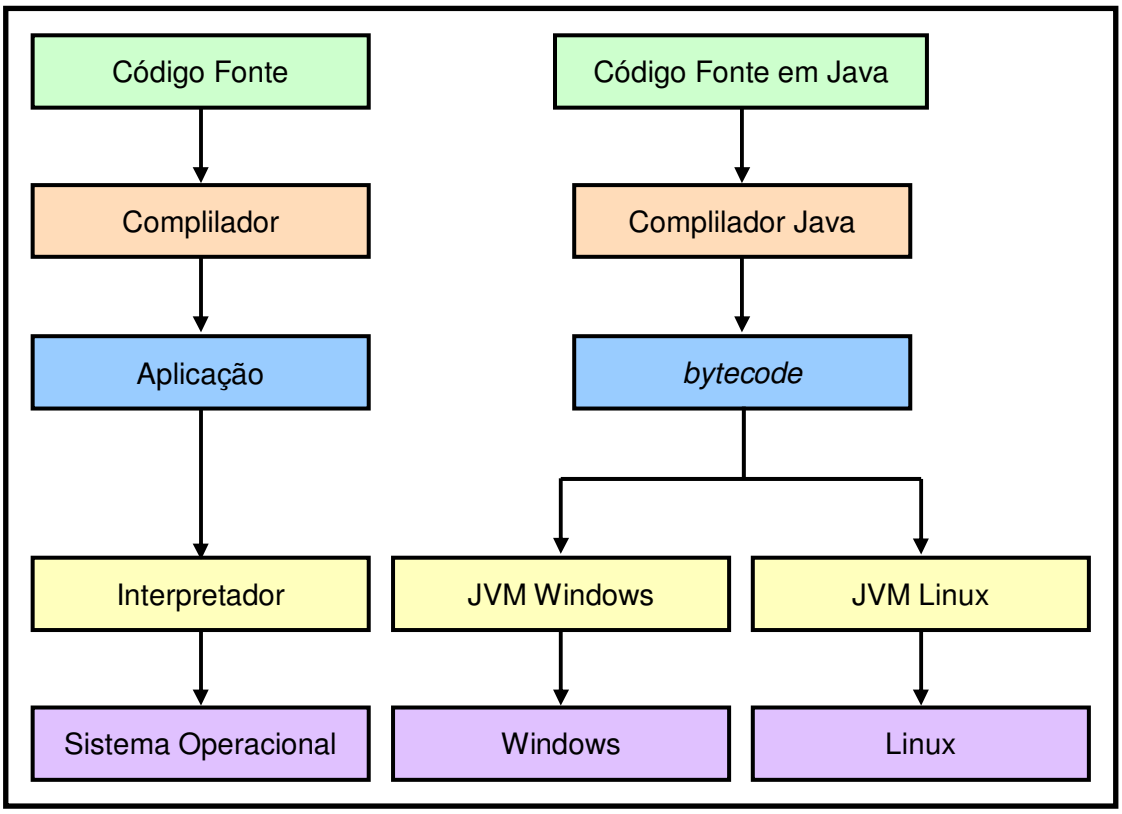


Figura 3: Arquitetura da Linguagem Java

Para que essa arquitetura funcione o Java ao compilar um código fonte cria um *bytecode*[6]. Este código não é legível aos olhos humanos, porém contém as diretrizes de execução que a JVM deve enviar ao sistema operacional.

Por isso, as aplicações escritas em linguagem Java são portáveis, ou seja, funcionam em diferentes SO, sem necessidade de reescrever ou compilar novamente o código.

Write once	Escreva uma vez
Run anywhere	Rode em qualquer lugar

Para que isto não cause problema de performance, a *Sun* desenvolveu a tecnologia Java *HOTSPOT* com o objetivo de maximizar o desempenho de programas executados em máquinas virtuais Java, isto é, verifica os pontos críticos do código e os otimiza.

[6] Estágio intermediário entre o código fonte e a aplicação final, uma linguagem pseudo-máquina que pode ser executado em qualquer JVM.

1.4. *Java - Componentes de uma Classe*

Uma classe Java é constituída por seu nome que delimita o seu escopo, onde temos também o escopo do método "*main*", área esta que abriga os atributos, comandos e outros métodos da classe. É importante o programador seguir as regras de endentação para garantir que os códigos estão entre os escopos delimitados por "{" para iniciar o escopo e "}" para finalizar o escopo, seja de uma classe ou método. Conforme segue abaixo:

```
public class <NomeDaClasse>                - abre o arquivo de classe
{                                          - indica o início da classe

    public  static void main (String  arg [])   - abre o início do método "main"

    {                                      - inicia o corpo do método

        System.out.print("Teste");         - corpo do método "main"

    }                                      -  indica o fim do método "main"
}                                          -  indica o fim da classe
```

Porém o compilador Java ignora mais de um espaço em branco entre as palavras.

A mesma classe escrita abaixo será compilada e executada corretamente, mas se houver algum erro na compilação, encontrá-lo será muito difícil, principalmente quando temos centenas de linhas de códigos.

```
public class Teste_I{ public static void main(String arg[]){ System.out.println ("A mesma coisa" ) ; } }
```

A organização do código é imprescindível para evitar erros e facilitar a manutenção dos mesmos, pois a classe abaixo é quase ilegível.

```
public class Teste_I{ public static void main(String arg[]){ System.out.println ("A mesma coisa" ) ; } }
```

Java é *case-sensitive,* ou seja, letras maiúsculas e minúsculas são distinguidas tanto no nome da classe quanto atributos, métodos e comandos:

Teste_I.java é diferente de **teste_I.java**

```
public class Teste_I
{
    public static void main(String arg[]){
        system.out.println ("System deve ser maiúscula" ) ;
    }
}
```

1.4.1. O método main

Para que uma classe possa ser executada no Java, é necessária a criação de um método principal, o método *main*.

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

Para entender melhor o funcionamento do Java, primeiro é importante saber o que é método. Segundo Filho (2005, p. 33) "Método em programação Orientada a Objetos é o nome que se dá ao conjunto de comando e palavras reservadas que determinam uma ação dentro de uma classe".

Quanto a estrutura do método *main*, começando pela palavra "***public***", esta notação indica que a classe pode ser usada por qualquer outra classe da aplicação, sendo assim é uma classe pública.

Já a palavra "***static***" é o identificador que diz que a implementação do método deve ser compartilhada com todas as instâncias[7] que forem feitas da classe original que o contém. Sendo assim é necessário alterar apenas uma vez a implementação do método para que todas as instâncias dessa classe original sejam alteradas ao mesmo tempo.

Para indicar que o método *main* não retornará nenhuma informação é utilizado a notação "***void***", indicando, portanto, que se trata de um método de retorno vazio.

A palavra "***main***" quer dizer principal. Desta forma, ao usá-la em um método, possibilita que esta classe seja executada. É a classe que representa o executável em analogia a linguagem C.

Os termos dentro dos parênteses "***String*[] *args***", representam a possibilidade do método receber argumentos, ou colocado de outra forma, valores em sua inicialização.

1.4.2. Comentários em Java

Na linguagem Java é possível fazer comentário somente de uma linha usando os símbolos "//" ou um bloco de informações iniciando com os símbolos "/*" e ao terminar os comentários finalize com "*/", conforme os exemplos a seguir:

[7] Instância é um processo utilizado em linguagem Orientada a Objetos para criar uma cópia de certa classe.

```
// Declaração do atributo idade
int idade;

/*
É necessário iniciar todos os atributos antes de
realizar as operações de soma.
*/
int idade = 10;
```

1.4.3. Erros

Erros freqüentes podem ser encontrados ao compilar um código. Demonstraremos alguns de acordo com o programa abaixo.

```
1       public class Exercicio_A_1 {
2               public static void main(String[] args) {
3                       System.out.println("Minha primeira linha de código!!!");
4               }
5       }
```

Um dos erros mais comuns é a falta do ; após os comandos. Veja que ele retorna à falta do ponto e vírgula (‘;’), mas aponta para a linha seguinte (4), pois ele entende que }é quem está finalizando o código, fique atento à esta peculiaridade.

Exercicio_A_1.java:4 ‘;’ expected

Não podemos declarar a classe com um nome e executá-la com outros como, por exemplo, executar a classe MeuPrg como meuprg. O resultado será o seguinte erro:

Exception in thread “main” Java.lang.NoClassDefFoundError: X(wrong name: x)

Esquecer a palavra static ou o argumento String [] args no método main, acarretará no erro:

Exception in thread “main” Java.lang.NoSuchMethodError: main

Não esqueça que o método main deve ser público, ou seja, deve iniciar com a anotação public, caso contrário ocorrerá o erro:

Main methods not public.

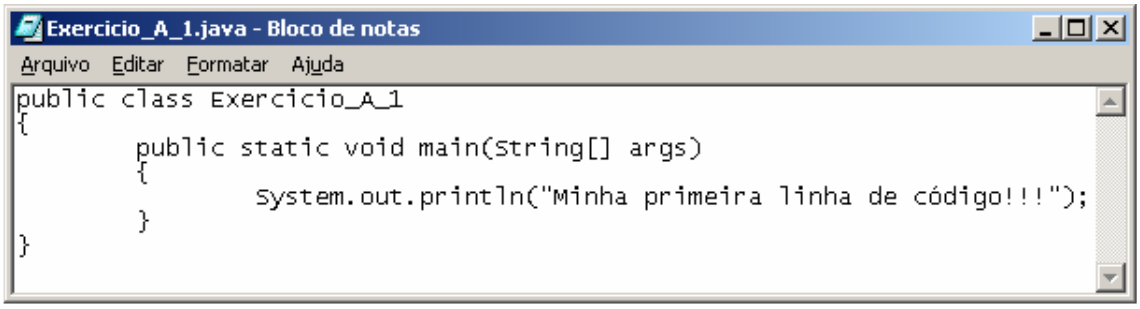
Esta lista de erros colabora na elaboração dos primeiros códigos, porém existem outros erros que acontecerão. Uma boa dica é anotar o erro e sua respectiva solução. Indica-se a reprodução dos erros citados acima para ganhar mais experiência.

1.5. *Java - Processo de Compilação*

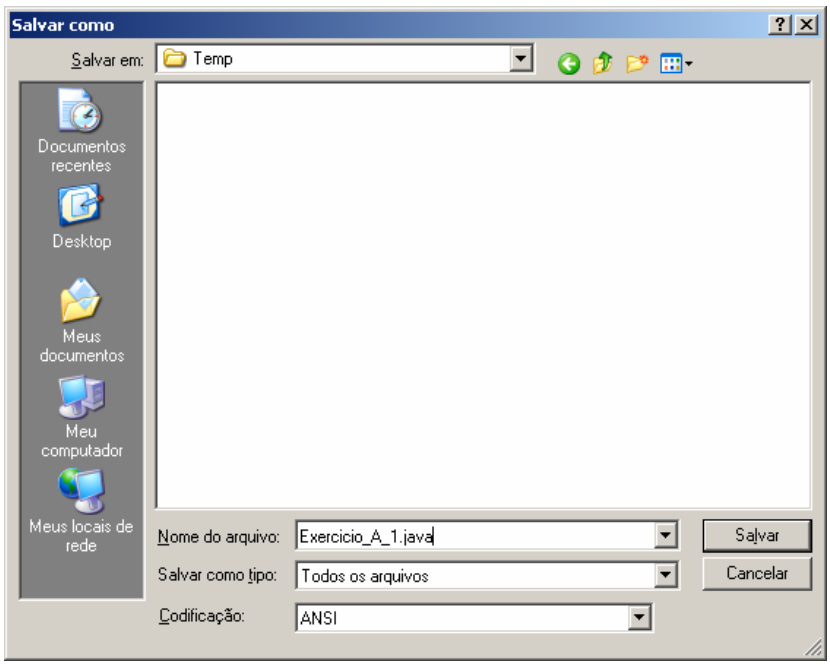
É importante lembrar que para compilar e executar uma classe Java é preciso instalar o JDK (instruções no anexo A).

Neste caso, será aproveitada a classe já utilizada como exemplo, com o objetivo de facilitar a compreensão do processo de compilação e execução.

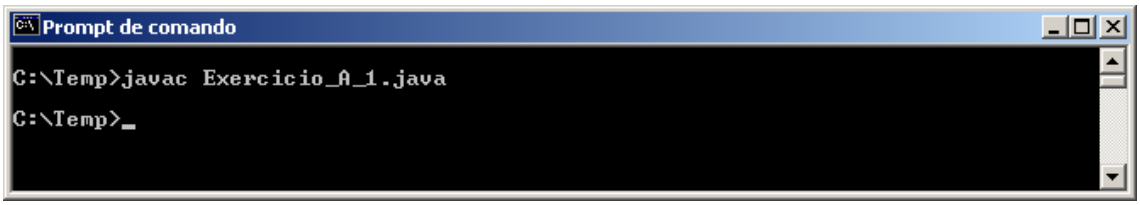
O primeiro passo é abrir um editor de texto, podendo ser o bloco de notas, e digitar a classe abaixo:



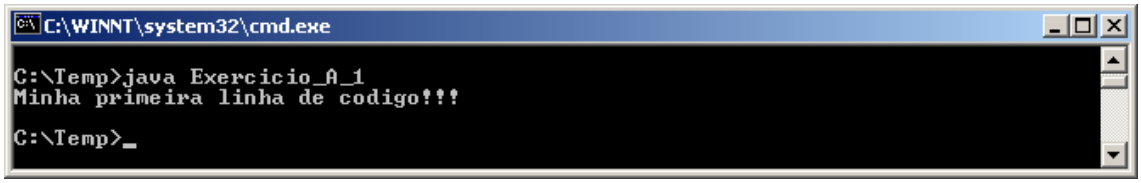
Salvar o arquivo com a extensão .java.



Iniciar o Prompt do DOS e no diretório que se encontra o arquivo Exercicio_A_1.java, digitar:



Se não ocorrer nenhum erro, nada será apresentado, então, deve-se digitar:



Este é o resultado da classe.

É importante perceber que a estrutura de compilação e execução sempre serão a mesma. O que será mudado é o nome da classe.

C:\javac <NomeDaClasse>.<Extensão>

C:\java <NomeDaClasse>

Nota-se que após compilar a classe ela pode ser executada com o comando java <NomeDaClasse> quantas vezes for necessário. Somente se houver uma alteração no código-fonte é que esta classe deve ser recompilada.

2. Dados

2.1. *Java - Tipos Numéricos*

Na tabela abaixo, estão os tamanhos de cada tipo primitivo do Java.

TIPO	*TAMANHO*
boolean	1 bit
byte	1 byte
short	2 bytes
char	2 bytes
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes

Em Java, toda variável tem um tipo que não pode ser mudado uma vez que declarado:

```
tipoDaVaríavel nomeDaVariável;
```

No que se refere à declaração, o tipo ***boolean*** armazena um valor verdadeiro ou falso. Sendo assim, representa apenas dois estados: true ou false.

```
boolean fim_do_arquivo = true;

boolean existe = false;
```

O tipo ***byte*** é um Inteiro de 8 bits em notação de complemento de dois. Pode assumir valores entre: -128 e 127.

```
byte a;
a = 125;

byte mes = 12;
```

O tipo ***short*** é um Inteiro de 16 bits em notação de complemento de dois. Os valores possíveis cobrem a faixa de: -32.768 e 32.767.

```
short b;
b = 32767;

short ano = 2008;
```

O tipo ***int*** é um Inteiro de 32 bits em notação de complemento de dois, o tipo mais usado em Java para números inteiros. Pode assumir valores entre: -2.147.483.648 e 2.147.483.647.

```
int c;
c = 253586;

int idade = 13;
```

O tipo ***long*** é um Inteiro de 64 bits em notação de complemento de dois. Podendo assumir valores entre -2^{63} e $2^{63}-1$.

```
long d;
d = 12500015;

long qtde = 100000354;
```

Utiliza-se o tipo ***float*** para números em notação de ponto flutuante normalizada em precisão simples de 32 bits onde o menor valor positivo representável é -9.223.372.036.854.775.808 e o maior é 9.223.372.036.854.775.807.

```
float e;
e = 125.00015;

float valor = 10000.0354;
```

O tipo ***Double*** é utilizado para números em notação de ponto flutuante normalizada em precisão dupla de 64, este também é um dos mais utilizados para representar os números fracionados. O menor valor positivo representável é -1.7976e+308 e o maior é 1.7976e+308

```
double f;
f = 3.14;

double Pi = 3.1415 926;
```

2.2. *Java - Tipos Textuais*

O tipo ***char*** guarda um e apenas um caractere e esse caractere deve estar entre aspas simples.

```
char g;
g = 'a';

char letra = 'k';
```

Esses tipos de variáveis são tipos primitivos do Java: o valor que elas guardam é o real conteúdo da variável.

Pode-se utilizar o tipo não primitivo ***String,*** para armazenar uma palavra ou uma frase. A representação de uma String se dá colocando os caracteres entre aspas dupla.

```
String h;
h = "Teste";

String frase = "Java é no GUJ";
```

2.3. *Java - Operadores Aritméticos*

É possível realizar uma série de operações com os dados do tipo inteiro. Inicia-se com as quatro operações básicas da matemática:

Operadores: +, -, *, /, %

Descrição: Adição, subtração, multiplicação, divisão e módulo (resto da divisão).

```
int idade = 5 + 1;
int idade = 5 - 1;
int idade = 5 * 1;

double valor = 5 / 2;
double valor = 5 % 2;
```

Teste essas variáveis e veja o resultado de cada uma.

```
class Exercicio_B_1{

	public static void main(String[] args) {

		int idade = 5 + 1;
		System.out.println("Idade + 1 => " + idade);

		int idade1 = 5 - 1;
		System.out.println("Idade - 1 => " + idade1);

		int idade2 = 5 * 1;
		System.out.println("Idade * 1 => " + idade2);

		double valor = 5.0 / 2;
		System.out.println("Valor / 2 => " + valor);

		double valor1 = 5.0 % 2;
		System.out.println("Valor % 2 => " + valor1);
	}
}
```

UNIP – Universidade Paulista
Laboratório de Prática de Programação

Os operadores de incremento e decremento referem-se a apenas uma variável. É uma forma objetiva de se escrever **x = x + 1**. Porém, esses operadores se comportam de modo diferente quando seguem ou precedem o nome de uma variável. Se o operador precede o nome da variável, então o incremento (ou decremento) ocorre antes que o valor da variável seja tomado para a expressão aritmética. Quando o operador segue o nome da variável o incremento ocorre depois que o valor da variável foi tomado pela expressão aritmética.

Operadores: ++, --

Descrição: Operadores de Incremento e decremento

x = x + 1; o mesmo que **x++;**
x = x - 1; o mesmo que **x--;**

```
int x = 5 ;
int y = x++;
```

x = x + 1; o mesmo que **++x;**
x = x - 1; o mesmo que **++x;**

```
int x = 5;
int y = ++x;
```

Teste essas variáveis e veja o resultado de cada uma.

```
class Exercicio_B_2{

        public static void main(String[] args) {

                int idade = 5;
                int nova_idade = idade++;
                System.out.println("Idade = > " + idade);
                System.out.println("Nova Idade = > " + nova_idade);

                int idade1 = 5;
                int nova_idade1 = ++idade1;
                System.out.println("Idade = > " + idade1);
                System.out.println("Mesma Idade = >"+ nova_idade1);

        }
}
```

UNIP – Universidade Paulista
Laboratório de Prática de Programação

Operadores: =, +=, -=, *=, /=, %=

Descrição: Operadores de atribuição.

x += 5 é o mesmo que **x = x + 5**
x -= y é o mesmo que **x = x - y**
x *= 2 é o mesmo que **x = x * 2**
z /= 4 é o mesmo que **z = z / 4**
w /= 4 é o mesmo que **w = w / 4**

```
int idade += 5;
int idade1 += 5;
int idade2 -= 5;
int idade3 *= 5;

double valor /= 5;
double valor1 %= 5;
```

Teste essas variáveis e veja o resultado de cada uma.

```
class Exercicio_B_3{

        public static void main(String[] args) {

                int idade = 5;
                System.out.println(“Idade = > ” + idade);

                int idade1 = 1;
                int idade1 += 5;
                System.out.println(“Idade += > ” + idade1);

                int idade2 = 1;
                int idade2 -= 5;
                System.out.println(“Idade -= > ” + idade2);

                int idade3 = 1;
                int idade3 *= 5;
                System.out.println(“Idade *= > ” + idade3);

                double valor = 2;
                valor /= 5.0;
                System.out.println(“Valor /= > ” + valor);

                double valor1 = 2;
                valor1 %= 5.0;
                System.out.println(“Valor1 %= > ” + valor1);
        }
}
```

2.4. *Java - Concatenação*

Tanto o tipo *char* quanto *String* são concatenados da mesma forma.

Operador: +

Descrição: Concatenação.

```
char vita = 'B';
char mina = '1';
char b12 = '2';

String estado = "São Paulo";
String pais = "Brasil";
```

Teste essas variáveis e veja o resultado de cada uma.

```
class Exercicio_B_4{

        public static void main(String[] args) {

                char vita = 'B';
                char mina = '1';
                char b12 = '2';

                System.out.println(vita + mina + b12);

                String estado = "São Paulo";
                String pais = "Brasil";
                System.out.println(estado + " - " + pais);
        }
}
```

3. Comandos de Decisão

Para que uma aplicação possa ser desenvolvida com sucesso é preciso analisar os pontos de decisão que a mesma tomará. Mas como fazer isso? Em primeiro lugar é importante se atentar para o fato de que isto ocorre o tempo todo em nossa vida: quando se compra um presente, primeiro verifica-se a quantia de dinheiro suficiente para adquiri-lo, correto? Diante desta verificação, se o há mais dinheiro do que custa o presente, então é possível comprá-lo, caso contrário será necessário escolher outro presente.

Este e outros tipos de decisões são tomadas todos os dias, quando se trata de programação não é diferente, deve-se identificar e analisar os pontos de decisão da aplicação com muita precisão, pois são estas decisões que guiaram os resultados da aplicação.

3.1. *Java – IF*

A forma mais simples de decisão de fluxo é o comando **if**. É empregado para executar condicionalmente comandos mediante um critério. Esse critério é dado por uma expressão, cujo valor resultante deve ser um dado do tipo *booleano*, isto é, **true** ou **false**. Se esse valor for **true**, então os comandos são executados; se **false**, a execução do programa segue adiante. A sintaxe para esse comando é:

```
if (condiçãoBooleana) {
        código;
}
```

```
int idade = 15;
if (idade == 18) {
        //Esta ordem não é executada, pois o valor do atributo idade não é igual a 18.
        System.out.println("Não pode entrar");
}
```

3.2. *Java – Operadores Relacionais*

Para escrever a condição do comando *if*, pode-se usar os operadores relacionais: **==, != , <, >, <=, >=** .

```
int idade = 15;
if (idade < 18) {
        //Esta ordem é executada, pois o valor do atributo idade é menor que 18.
        System.out.println("Não pode entrar");
}

int idade = 15;
if (idade > 18) {
        //Esta ordem não é executada, pois o valor do atributo idade não é maior que 18.
        System.out.println("Não pode entrar");
}

int idade = 15;
if (idade != 18) {
        //Esta ordem é executada, pois o valor do atributo idade é diferente de 18.
        System.out.println("Não pode entrar");
}
```

Faça o mesmo com os outros operadores e veja o resultado.

3.3. *Java – Operadores Lógicos*

Considere-se a seguinte situação: você tem 15 anos e está acompanhado de um amigo que tem 18 anos, logo você pode entrar caso esteja acompanhado deste amigo. Então você **OU** seu amigo devem ter idade igual ou maior que 18 anos.

Na maioria dos casos de tomada de decisão, precisamos do auxilio dos operadores lógicos, ***And***(e) e ***OR***(ou)que são representados por &&, || respectivamente.

```
int sua_idade = 15;
int idade_amigo = 19;
if (sua_idade > 18 || idade_amigo > 18) {
        System.out.println(“Podem entrar”);
}
```

No caso abaixo, você não poderia entrar, pois a condição exige que os dois tenham mais de 18 anos.

```
int sua_idade = 15;
int idade_amigo = 19;
if (sua_idade > 18 && idade_amigo > 18) {
        System.out.println(“Podem entrar”);
}
```

3.4. *Java – Operador Textual*

Mais um detalhe está relacionado ao tipo String, que se utiliza do operador ***equals*** e da negação "**!**" para fazer comparações conforme o exemplo abaixo:

```
int sua_idade = 15;
String nome_amigo = “Luis”;
if (sua_idade > 18 || nome_amigo.equals(“Luis”) ) {
        System.out.println(“Podem entrar”);
}

int sua_idade = 15;
String nome_amigo = “Luisa”;
if (sua_idade > 18 || !nome_amigo.equals(“Luis”) ) {
        System.out.println(“Podem entrar”);
}
```

3.5. *Java – ELSE*

Além disso, pode-se usar a cláusula ***else*** para indicar o comportamento que deve ser executado no caso da condição ser falsa: Como se estivesse dizendo: Se não pode fazer isto, então, faça aquilo.

```
if (condiçãoBooleana) {
        código;
}
else {
        código
}

int idade = 15;
if (idade > 18) {
        System.out.println(“Pode entrar”);
}
else {
        System.out.println(“Não pode entrar”);
}
```

Freqüentemente, deseja-se que um único bloco de comandos de uma lista seja executado mediante um dado critério. Isso pode ser feito através do encadeamento ou acoplamento de vários comandos **if-else**, do seguinte modo:

```
if (condiçãoBooleana) {
        código;
}
else if (condiçãoBooleana) {
        código;
}
else {
        código;
}

/* Desejamos definir y tal que
          | x+2,  se x < -1,
      y = | 1,    se -1 <= x < 1,
          | x*x,  se 1 <= x.
*/
if (x < -1){
        y = x+2;
}
else if(x <= 0){
        y = 1;
else {
        y = x*x;
}
```

3.6. *Java – SWICH, CASE*

Da mesma forma que *IF*, o ***swich*** ajuda na tomada de decisão, porém deve-se usar apenas quando apenas uma entre as condições testadas é verdadeira. Se o valor for diferente de todas essas constantes, então o comando presente sob o rótulo **default:** é executado, caso este esteja presente. Por exemplo:

```
switch ( condição ) {
        case opção1:
                código;
        case opção2:
                código;
        case opçãoN:
                código;
        default:
                código;
}

int op;
op = 2;

switch(op) {
        case 1:
                System.out.println("case 1: op=" + op);
                break;
        case 2:
                System.out.println("case 2: op=" + op);
                break;
        case 3:
                System.out.println("case 3" + op);
                break;
        default:
                System.out.println("default: op não está no limite 1..3");
                break;
   }
```

4. Comandos de Repetição

Quando é preciso repetir incondicionalmente um ou mais comandos um determinado número de vezes para se obter um resultado, deve-se usar os comandos: for, while e do while

4.1. *Java - For*

Em certas situações é preciso laços de repetições nos qual alguma variável é usada para contar o número de iterações. Para essa finalidade, há o laço ***for***. É a estrutura de repetição mais utilizada nas aplicações. Destaca-se pela inicialização, condição e incremento, logo no início da sua estrutura.

```
for (inicialização; condição; incremento) {
        código;
}
```

```
for (int i = 0; i < 10; i++) {
        System.out.println(“olá!”);
}
```

O código do for indica claramente que a variável i serve em especial para controlar a quantidade de laços executados.

4.2. *Java - While*

A estrutura de repetição ***while*** repete-se, ou seja, forma um *loop*, enquanto sua condição estiver satisfeita, sendo assim, enquanto a condição for verdadeira continua-se repetindo.

```
inicialização;
while(condição) {
        código;
        incremento;
}
```

```
int idade = 15;
while( idade < 18 ) {
        System.out.println( idade );
        idade++;
}
```

O trecho dentro do bloco do *while* será executado até o momento em que a condição idade < 18 passe a ser falsa.

Uma das observações importantes é sempre certificar que não ocorra o laço infinito já que ele não terminará enquanto a condição for verdadeira.

4.3. *Java – Do While*

Já o ***do while*** sempre será executado pelo menos uma vez e então fará a verificação da condição. Dessa maneira diz-se: faça enquanto a condição for verdadeira.

```
do {
        código;
}
while( condição );
```

```
int idade = 18;
do {
        System.out.println( idade );
        idade++;
}
while( idade < 18 );
```

Diferente do *while*, este tipo de laço de repetição executa o código e em seguida avalia a expressão condicional. A repetição ocorre se o valor dessa expressão for verdadeira. Sendo assim, o código acima representaria um problema, verifique e resolva-o.

5. Comandos de Controle e Arrays

5.1. *Java - Break*

Em certos casos é preciso interromper a execução de um determinado bloco de comando, para tal finalidade utiliza-se o comando ***break***.

```
for (int i = x; i < y; i ++) {
        if (i % 19 == 0) {
                System.out.println("Achei um número divisível por 19 entre x e y");
                break;
        }
}
```

5.2. *Java - Continue*

Da mesma maneira, é possível obrigar o loop a executar o próximo laço. Para isso utiliza-se o comando ***continue***.

```
for (int i = 0; i < 100; i++) {
        if( i > 50 && i < 60) {
                continue;
        }
System.out.println( i );
}
```

O código acima não vai imprimir alguns números. Faça este teste.

5.3. *Java - Array*

Até agora para usar vários atributos de mesmo tipo, era necessário que cada um tenha um nome. Por exemplo:

```
int idade1;
int idade2;
int idade3;
int idade4;
```

Agora se deve declarar uma **matriz** (***array***) de inteiros:

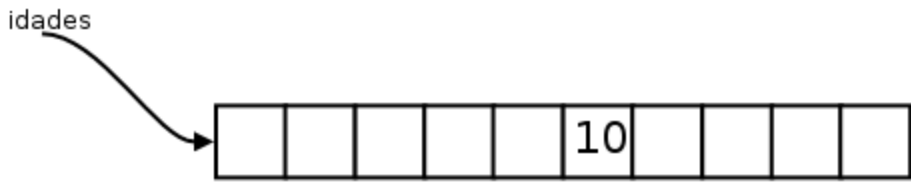
```
int[] idades;
```

Uma *array* é sempre um objeto, sendo assim, o atributo idades é uma referência. É preciso criar um objeto para usar a array.

```
idades = new int[10];
```

É criada uma array de 10 posições e logo abaixo é atribuído um valor ao seu endereço.

```
idades[5] = 10;
```



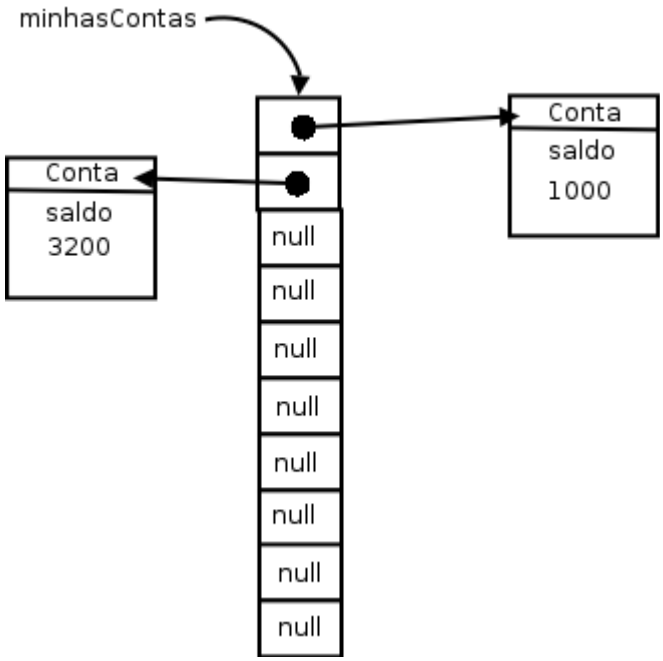
Veja que a posição atribuída foi a sexta, pois o Java os índices do *array* vão de 0 a n. Onde n é o fim do tamanho de sua *array*.

Como *array* são apenas referências, quando é criada ela apenas tem seus espaços vazios a espera de valores, conforme código abaixo:

```
Conta contaNova = new Conta();
contaNova.saldo = 1000.0;
minhasContas[0] = contaNova;
```

Ou pode-se atribuir o valor diretamente:

```
minhasContas[1] = new Conta();
minhasContas[1].saldo = 3200.0;
```



5.3.1. Percorrendo uma Array

Para percorrer um arrai o comando mais adequando é o for, que aprendemos anteriormente:

```
1.      public static void main(String args[]) {
2.              int[] idades = new int[10];
3.              for (int i = 0; i < 10; i++) {
4.                      idades[i] = i * 10;
5.              }
6.              for (int i = 0; i < 10; i++) {
7.                      System.out.println(idades[i]);
8.              }
9.      }
```

Em casos que não sabemos o tamanho do array é utilizado o comando ***length*** Conforme o exemplo abaixo:

```
1.      void imprimeArray(int[] array) {
2.              for (int i = 0; i < array.length; i++) {
3.                      System.out.println(array[i]);
4.              }
5.      }
```

Após a criação de uma *array* seu tamanho não poderá ser mudado, caso precise de mais espaço é necessário a criação de uma nova *array*.

6. Referências Bibliográficas

- DEITEL, H. M. **Java Como Programar**. Porto Alegre: Bookman, 2005.

- FILHO, Renato Rodrigues. **Desenvolva aplicativos com Java 2**. São Paulo: Érica, 2005.

- HORSTMANN, Cay. **Conceitos de Computação com o Essencial de Java.** 3.ed. Porto Alegre: Bookman, 2005.

- SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java**. Rio de Janeiro: Campus, 2003.

- CAELUM – ENSINO E SOLUÇÕES EM JAVA. **Java e Orientação a Objetos**. Disponível em: <http://www.caelum.com.br/caelum/curso-11.jsp>. Acesso em: 30 jan. 2008, 15:02.

- CAELUM – ENSINO E SOLUÇÕES EM JAVA. **Java para desenvolvimento Web**. Disponível em: <http://www.caelum.com.br/caelum/curso-21.jsp>. Acesso em: 30 jan. 2008, 16:20.

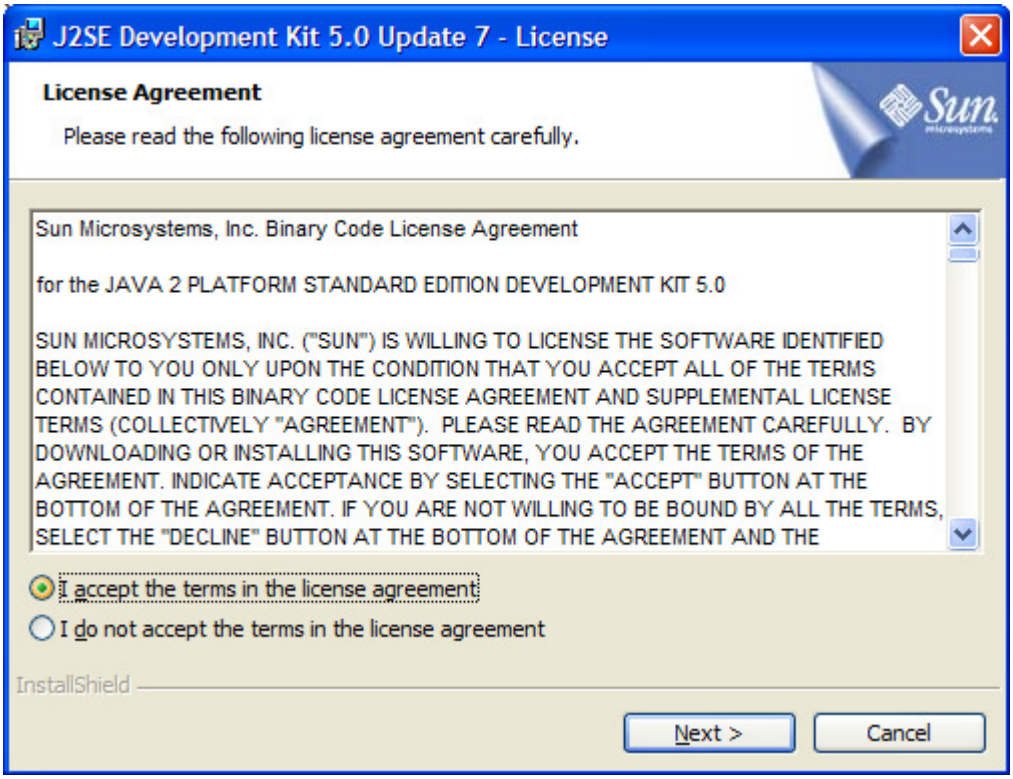
7. Anexo A

7.1. *Instalação do JDK em Ambiente Windows*

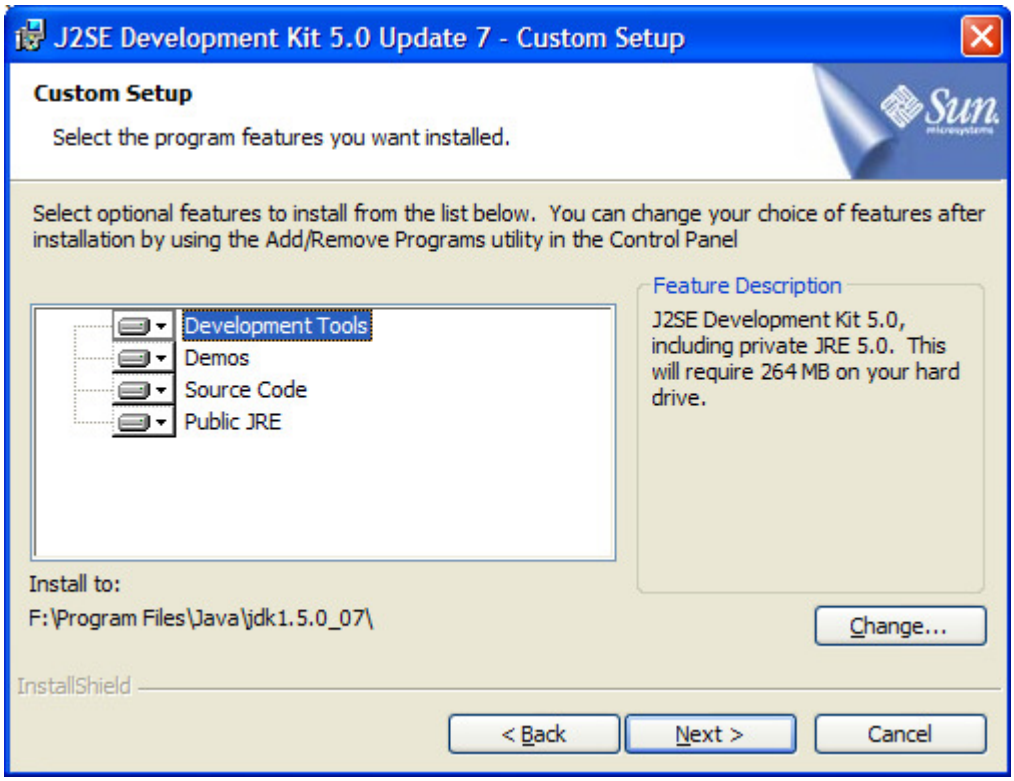
Para instalar o JDK no Windows, primeiro baixe-o no site da *Sun*, é um simples arquivo executável, que contém o Wizard de instalação.

1) Dê um clique duplo no arquivo de instalação e espere até entrar no wizard de instalação.

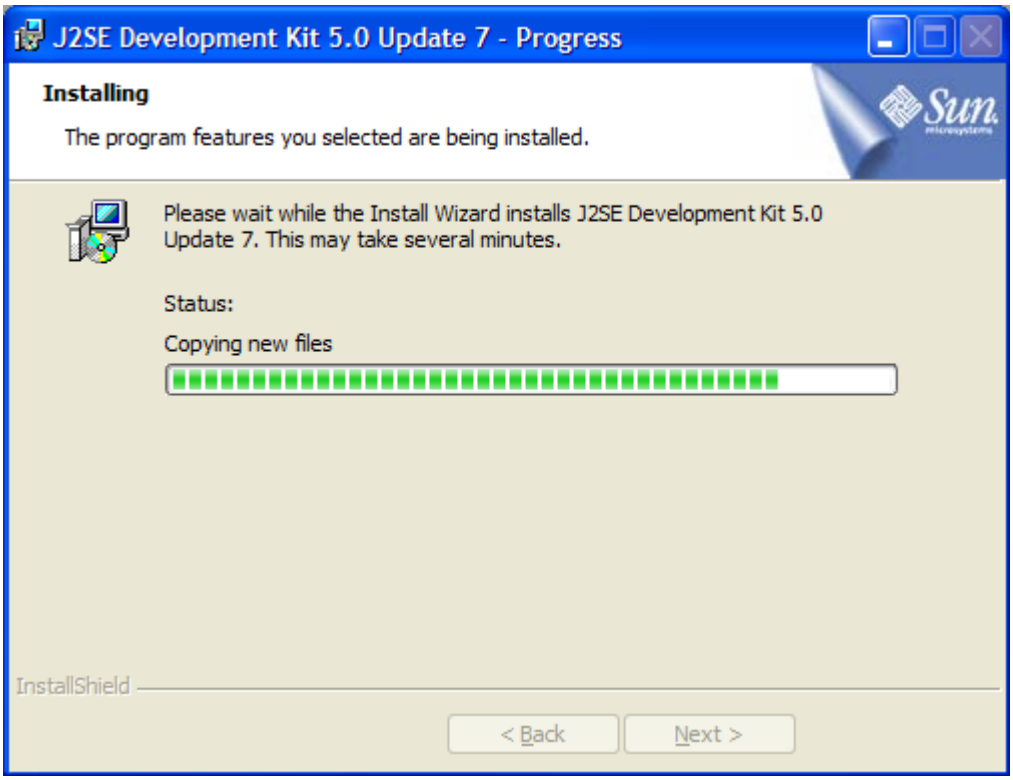
2) Neste tela aceite o contrato da *Sun*, marcando a opção "*I accept the terms in the license agreement*" e clique em "*Next*".



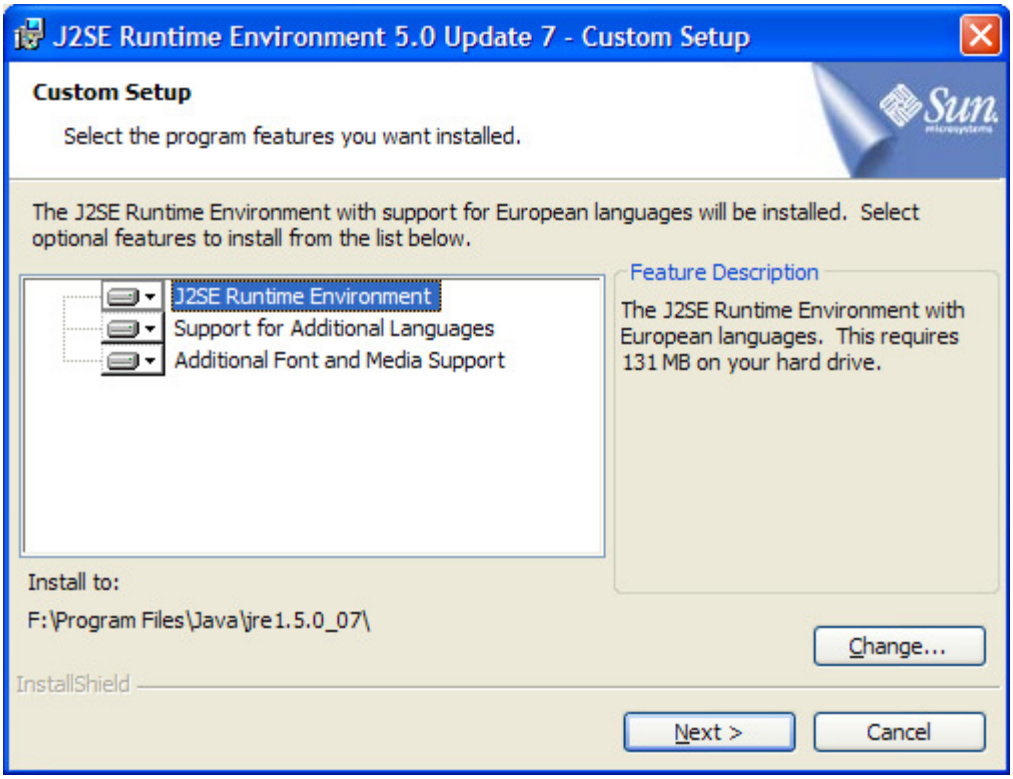
3) Agora devemos selecionar quais recursos instalaremos junto com o java (Ferramentas de desenvolvimento, Demonstrações, o código fonte e o próprio java), e onde ele será instalado (marque esse caminho porque usaremos ele mais pra frente), deixe como está e clique em "*Next*".



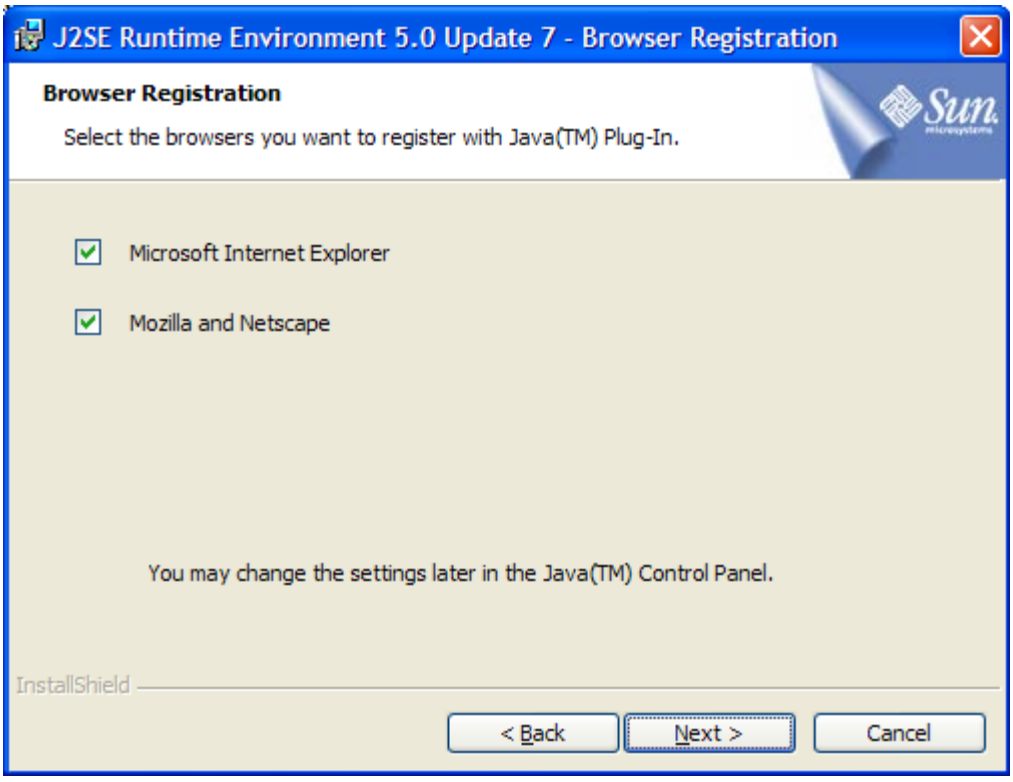
4) Pronto, agora ele começará a instalar o JDK !



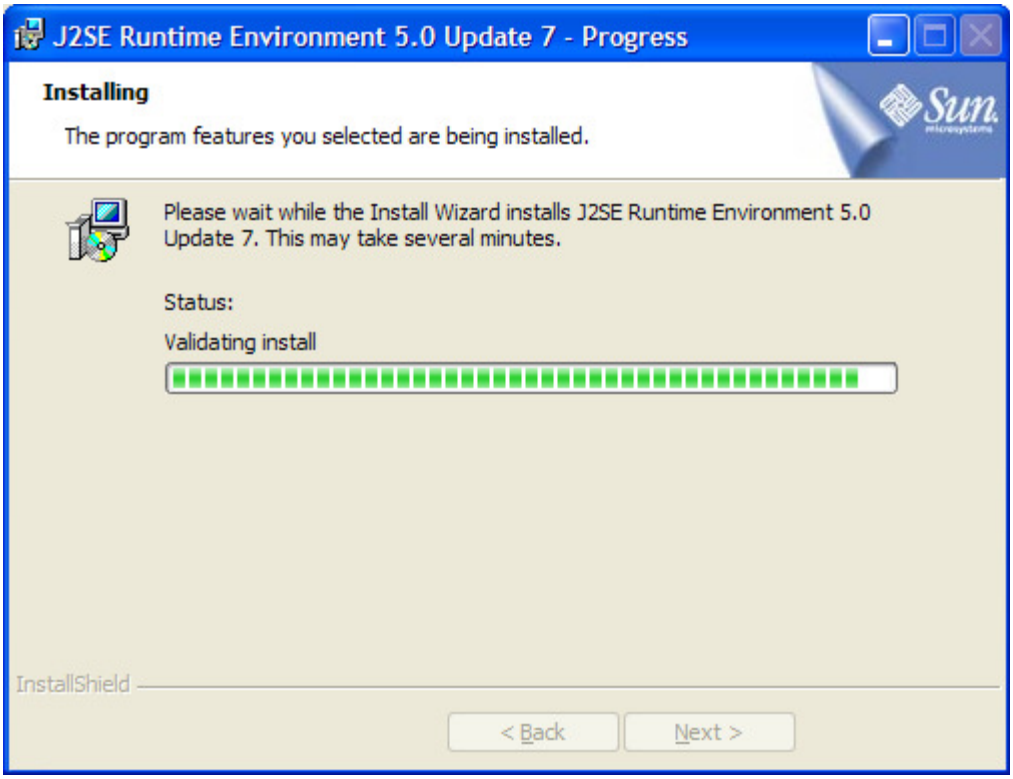
5) Agora ele começará a instalar o JRE (*Java Runtime Environment*). Assim como o JDK, ele também tem algumas opções. Deixe como está e clique em "*Next*".



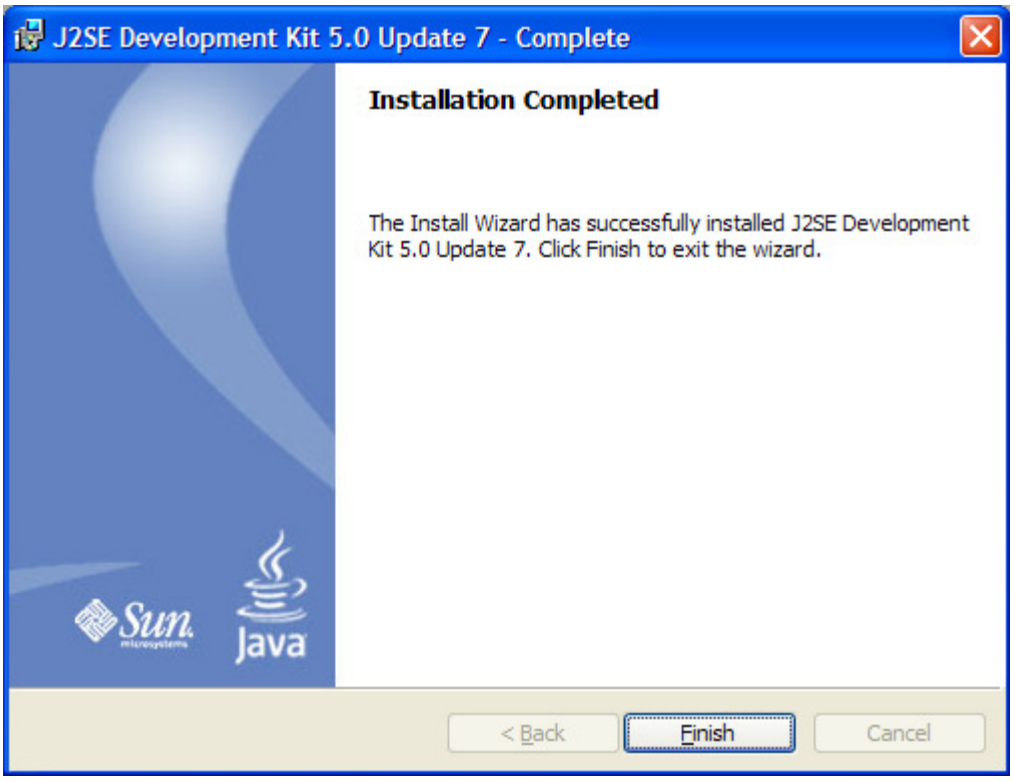
6) Neste passo, você configura os navegadores para utilizaram o Java, por exemplo pra rodar um applet.



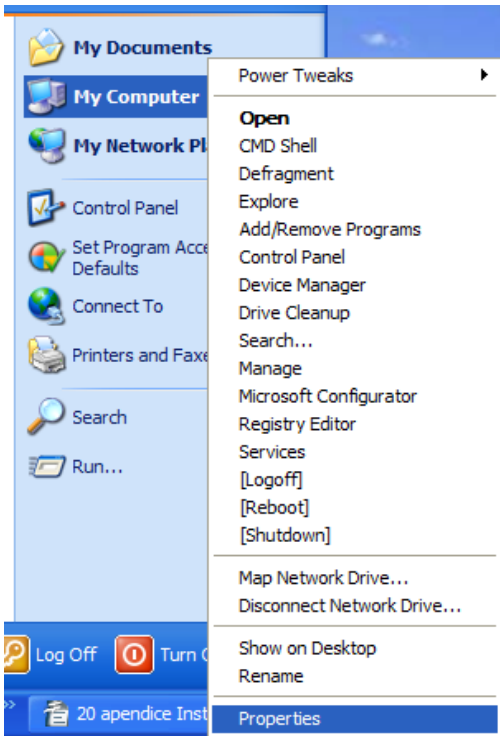
7) Pronto agora ele instalará o JRE.



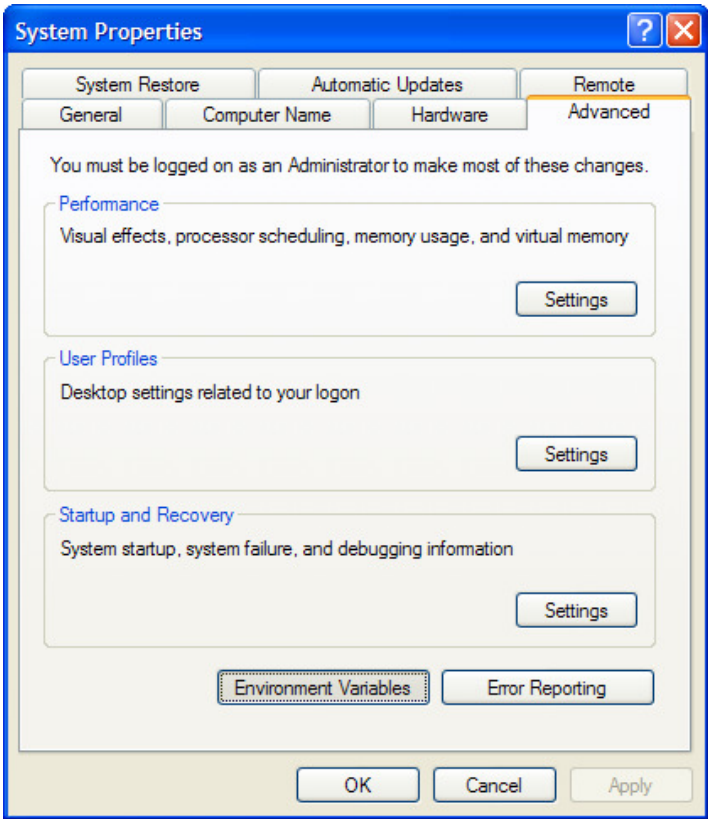
8) Agora seu JDK está instalado. Clique em Finish.



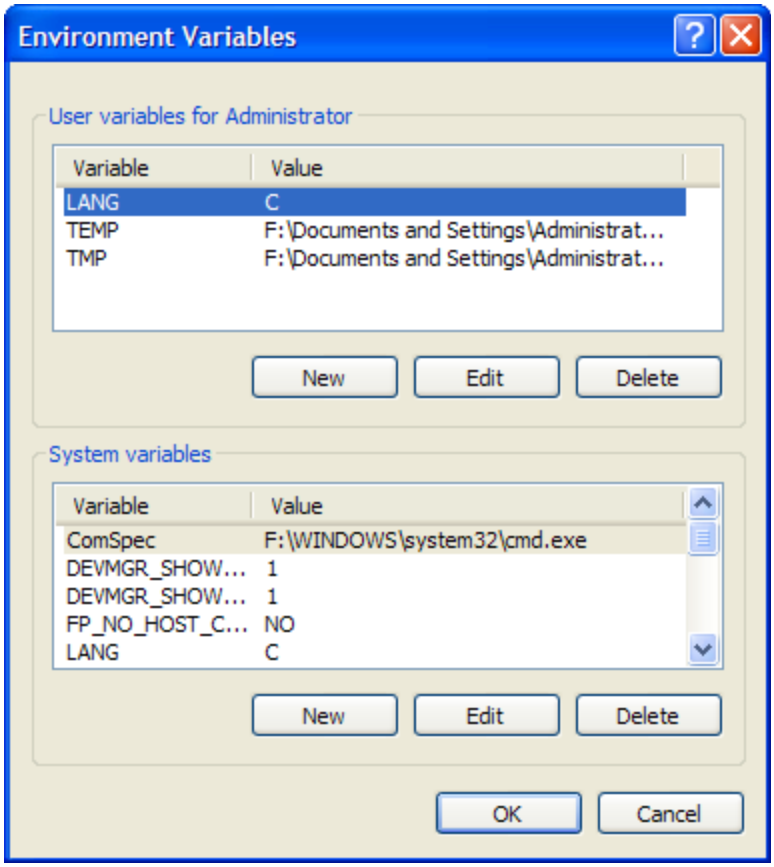
9) Agora vamos criar as variáveis de ambiente. Clique com o botão direito em cima do ícone “Meu Computador e selecione a opção “Propriedades”.



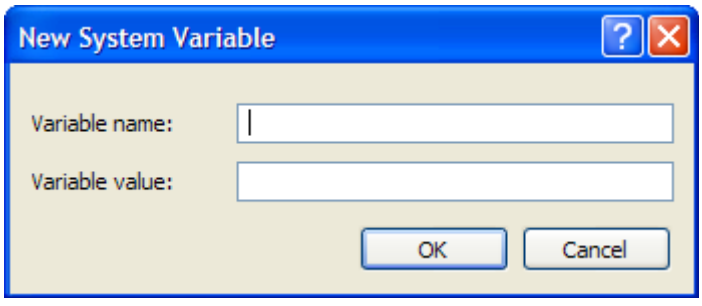
10) Agora escolha a aba “Avançado” e depois clique no botão “Variáveis de Ambiente”



11) Nesta tela você verá na parte de cima, as variáveis de ambiente do usuário corrente , e abaixo, as variáveis de ambiente do computador (serve para todos os usuários). Clique no botão "*New*" da parte de baixo

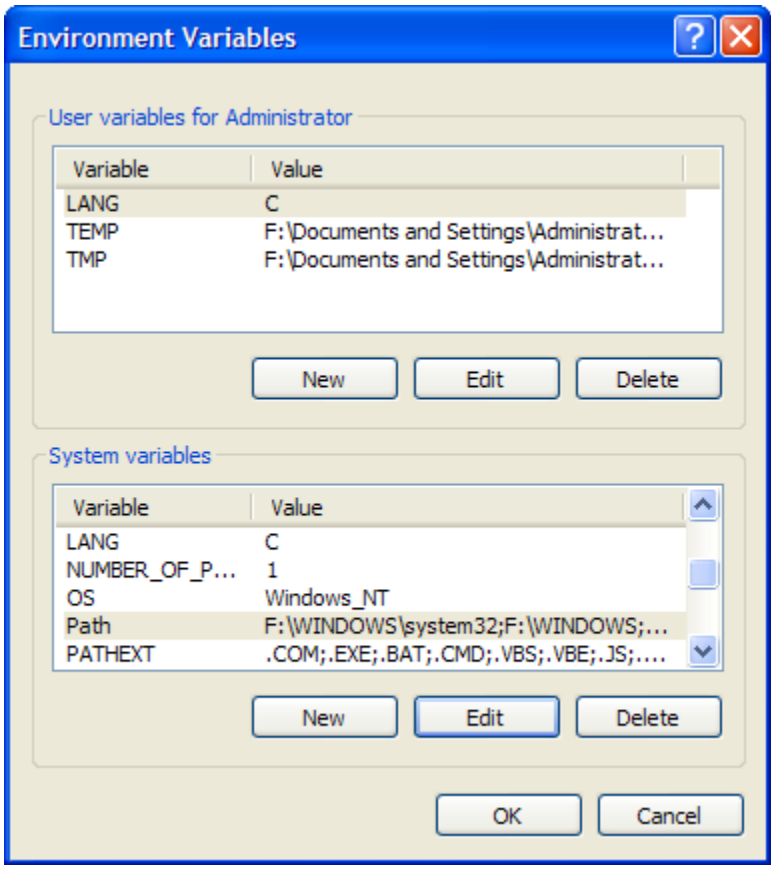


12) Agora em "Nome da Variável" digite JAVA_HOME, e em valor da variável digite o caminho que você anotou no passo 3. Nesta máquina o caminho é F:\Program Files\Java\jdk1.5.0_07\, mas na sua máquina provavelmente vai ser outro como "C:\Arquivos de Programas\Java\jdk1.5.0_07\". E depois clique em OK.



13) Crie uma nova variável de ambiente repetindo o passo 11, porém agora defina o nome da variável como CLASSPATH e o valor com . (ponto).

14) Agora não vamos criar outra variável e sim alterar, para isso procure a variável PATH, ou Path e clique no botão de baixo “Editar”.



15) Não mexa no nome da variável, deixe como está, e adicione no final do valor ;%JAVA_HOME%\bin, não esqueça do ponto e vírgula, assim você está adicionando mais um caminho à sua variável Path.

16) Agora abra o *prompt* e digite *javac* -version se mostrar a versão do Java *Compiler* e algumas opções, caso não apareça reveja os passos e confira se não esqueceu ou pulou nenhum deles.