Desenvolvimento de Software

Aula 01: Características gerais e sintaxe básica

Apresentação

A linguagem Java teve origem na década de 1990 e de lá para cá foi se tornando cada vez mais popular, sendo amplamente utilizada em sistemas críticos.

Como toda linguagem, esta também apresenta peculiaridades, como o uso de padrão de escrita CamelCase, paradigma orientado a objetos e processamento paralelo nativo.

Precisamos conhecer os tipos utilizados, operadores e controles de fluxo, dando assim os primeiros passos na linguagem, e o uso de um ambiente de desenvolvimento integrado é altamente recomendável para o ganho de produtividade.

Objetivos

- Identificar as características gerais do ambiente Java;
- Explicar a sintaxe básica da linguagem Java;
- Aplicar os elementos da sintaxe Java a diferentes contextos.

Características Gerais do Java

Java é uma linguagem de programação **orientada a objetos** desenvolvida na década de 1990 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Atualmente, a linguagem está sob o controle da Oracle, após ter comprado a Sun.

Desde a sua concepção, o Java trouxe alguns princípios básicos que norteiam a sua própria evolução, destacando-se o cuidado em executar em **múltiplas plataformas** e garantir elementos de **conectividade**.

Em linguagens que geram código nativo para o Sistema Operacional, como C++ e Pascal, o código fonte deve ser compilado, gerando arquivos objeto, e estes arquivos devem ser combinados por **linkedição** para criar um executável.

Outras linguagens, classificadas como **interpretadas**, não passam por este processo de compilação, sendo executadas linha a linha por um interpretador, a exemplo da programação shell para Unix e do JavaScript nos navegadores.

Atenção

Aqui cabe uma observação de que Java e JavaScript não podem ser confundidos, tratando de tecnologias totalmente distintas.

A linguagem Java utiliza um artifício que permite a execução de seus programas em qualquer plataforma: uso de **máquina virtual**. Com esta abordagem, os programas em Java são compilados, não podendo ser classificados como interpretados, mas sem gerar executáveis para o Sistema Operacional, logo não sendo linkeditados.

Cada sistema operacional tem a sua versão da máquina virtual Java, normalmente necessitando da instalação do Java Runtime Environment (JRE), disponível gratuitamente no site da Oracle, e os programas feitos em Java irão executar nestas máquinas virtuais, sem se preocupar com o sistema hospedeiro.

Isto confere à linguagem a característica de ser **conceitualmente multiplataforma**. Na prática, os programas executam em apenas uma plataforma, a máquina virtual, que assume o papel de "player" de Java, funcionando como os demais "players" para filmes e músicas.

Nós também temos uma grande facilidade para efetuar a conexão com outros sistemas em rede com o uso de Java, por que traz uma extensa biblioteca de componentes para a criação de **Sockets**, conexão **HTTP** e **FTP**, criação de clientes e servidores de e-mail, entre diversos outros, garantindo a premissa básica de conectividade da linguagem.

Outra característica inovadora no lançamento do Java, e rapidamente adotada por outras linguagens, foi a inclusão de um coletor de lixo (**garbage collector**), o qual efetua a desalocação de memória de forma automática.

Em termos de programação, devemos estar atentos ao fato de que o Java é **fortemente tipado**, ou seja, o tipo da variável é definido na sua declaração, além de ser **case sensitive**, diferenciando letras minúsculas e maiúsculas, e adotar o padrão de nomenclatura **CamelCase**.

Ambiente de Desenvolvimento

Vídeo.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Podemos desenvolver programas em Java com o simples uso de um editor de texto e os programas de compilação e execução oferecidos a partir do **Java Development Kit** (JDK), o qual pode ser obtido gratuitamente no site da Oracle (www.oracle.com).

Exemplo

Poderíamos criar um pequeno programa de exemplo no bloco de notas.

Você deve salvar este programa com o nome "Exemplo001.java", depois ir até o prompt e digitar dois comandos, no mesmo diretório onde o arquivo foi salvo, lembrando que o diretório bin do JDK deve estar no PATH.

```
javac Exemplo001.java
java Exemplo001
```

Nós utilizamos o primeiro comando para compilar o arquivo de código-fonte ".java" e gerar o arquivo-objeto ".class", enquanto o segundo comando executa este ".class", efetuando a chamada ao método main, ponto inicial de execução, e imprimindo a mensagem "Alô Mundo".

Note que as linhas de código em Java são terminadas com o uso de ponto e vírgula, e o esquecimento deste terminador é uma fonte de erros muito comum para os programadores da linguagem.

Apesar de ser possível trabalhar apenas com o JDK e o bloco de notas, precisamos de um ambiente que garanta produtividade nas tarefas que envolvem a programação, e para tal devemos adotar um ambiente integrado de desenvolvimento (IDE).

Várias IDEs estão disponíveis para Java, algumas gratuitas e outras pagas, mas todas elas precisam da instalação do JDK antes que as mesmas sejam instaladas.

Alguns exemplos de IDEs para Java:

1

Eclipse

2

NetBeans

3

IntelliJ IDEA

4

BlueJ

5

JCreator

Nós iremos adotar o NetBeans, por ser mais didático e já trazer uma configuração completa, sem a necessidade de adicionar servidores ou bancos de dados, o que trará mais conforto para todos na aprendizagem da linguagem.

A interface do NetBeans é bastante complexa, e devemos entender seus componentes principais para melhor utilização da ferramenta.

De acordo com a numeração utilizada na figura, temos os seguintes componentes:

1 **Menu Principal** – Controle global das diversas opções da IDE, ativação e desativação de painéis internos, instalação de plataformas, entre outras funcionalidades. Este é o controle de mais alto nível do NetBeans.

2 **Toolbar** – Acesso rápido, de forma gráfica, às opções mais utilizadas, como criar arquivos e projetos, salvar arquivos e executar o projeto.

3 **Painel de Controle** – Aceita várias configurações, mas no padrão normal apresenta uma divisão com a visão lógica dos projetos (**Projetos**), uma com a visão física (**Arquivos**) e outra com acesso ao banco de dados e aos servidores (**Serviços**).

4 **Navegador** – Permite o acompanhamento das características de qualquer elemento selecionado da IDE, como classes em meio ao código.

5 **Editor de Código** – Voltado para a edição do código, com diversos auxílios visuais e ferramentais de complementação.

6 **Saída** – Simula o prompt do sistema, permitindo observar a saída da execução, bem como efetuar entrada de dados via teclado.

O NetBeans pode ser obtido em **netbeans.org/downloads**, devendo ser escolhida a versão completa, pois já traz todas as plataformas de programação configuradas, além dos servidores GlassFish e Tomcat embutidos.

Após instalar o JDK e o NetBeans (verão completa), podemos executar esta IDE e criar o nosso primeiro programa já no ambiente de desenvolvimento completo.

Para criar um novo projeto devemos seguir os passos:

1. No menu principal do NetBeans escolha a opção Arquivo..Novo Projeto, ou Ctrl+Shift+N;

2. Na janela que se abrirá, escolha o tipo de projeto como Java..Aplicação Java e clique em Próximo;

3. Dê um nome para o projeto (Exemplo001) e diretório para armazenar seus arquivos (C:\MeusTestes) e clique em Finalizar.

Ao final destes passos, o projeto estará criado e sua estrutura poderá ser observada no **Painel de Controle**, bem como o código padrão estará presente no **Editor de Código**. Diversos comentários são adicionados a esse código, aparecendo normalmente na coloração cinza, e eles podem ser mantidos ou removidos sem qualquer interferência sobre a funcionalidade do projeto.

O que precisamos fazer agora é complementar esse código com a linha que falta no **main**, conforme pode ser observado a seguir.

```
package exemplo001;
public class Exemplo001 {
  public static void main(String[] args) {
    System.out.println("APENAS UM EXEMPLO");
  } }
```

Após acrescentar a linha com o comando de impressão, basta executar o projeto com uso de **F6**, menu **Executar..Executar Projeto**, ou botão “play” da **ToolBar** ( )

O projeto será compilado e executado, e a frase “APENAS UM EXEMPLO” poderá ser observada na divisão de **Saída** do NetBeans.

Atenção

Para todos os projetos de exemplo que criaremos a partir daqui teremos de seguir esses passos. Ocorrerão alterações apenas para outros tipos de projeto, como aplicativos web, e essas mudanças serão apresentadas no momento oportuno.

Tipos Nativos e Operadores

Quase tudo em Java é baseado em objetos. Basicamente, apenas os tipos nativos são considerados de forma diferente, mas para cada tipo nativo existe uma ou mais classes **Wrapper**.

Sempre devemos lembrar que um tipo nativo corresponde a uma simples posição de memória, enquanto uma classe Wrapper, além de comportar o valor correspondente ao tipo, oferece métodos específicos para tratamento do mesmo.

Podemos observar, no quadro seguinte, os principais tipos nativos do Java.

Tipo Nativo	Wrapper	Descrição do Tipo
byte	Byte	Inteiro de 1 byte
short	Short	Inteiro de 2 bytes
int	Integer	Inteiro de 4 bytes
long	Long	Inteiro de 8 bytes
char	Character	Caracteres ASCII
float	Float	Real de 4 bytes
double	Double	Real de 8 bytes
boolean	Boolean	Valores booleanos true ou false

As variáveis aceitam diferentes formas de declaração e inicialização, como podemos observar a seguir:

```
int a, b, c;
boolean x = true; // Variável declarada e inicializada
char letra = 'W';
String frase = "Teste";
double f = 2.5, g = 3.7;
```

Para programas simples não há necessidade do uso extensivo de classes Wrapper, mas é aconselhável utilizá-las na criação de sistemas que usem tecnologias de Web Service, ou frameworks de persistência, entre outros.

Atenção

Nas versões antigas do Java as classes Wrapper deveriam ser alocadas com uso de **new**, mas atualmente podem ser associadas direto ao valor armazenado pelas classes. Exceção feita à classe String, que sempre permitiu a inicialização direta.

Lembre-se sempre de comentar seu código para que você e outros programadores possam revisá-lo com mais facilidade em adaptações posteriores.

O Java aceita dois tipos de comentários:

Comentário de linha, com uso de //.

Comentário longo, iniciado com /* e finalizado com */.

Tendo definidas as nossas variáveis, podemos efetuar as mais diversas operações sobre elas com o uso de operadores. Os principais operadores são divididos em aritméticos, relacionais e lógicos.

Os operadores aritméticos utilizados no Java são:

Operador	Operação
+	Soma (concatenação para texto)
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento de 1
--	Decremento de 1
&	And (E) binário
|	Or (ou) binário
^	Xor (ou-exclusivo) binário
Operador	Operação

Vamos observar, a seguir, um pequeno programa de exemplo com o uso de operadores.

```
package exemplo002;

public class Exemplo002 {
  public static void main(String[] args) {
    int a = 5, b = 32, c = 7;
    System.out.printf("A: %d\t B: %d\t C:%d\n",a,b,c);
    b = b - c;
    b /= a;
    System.out.printf("A: %d\t B: %d\t C:%d\n",a,b,c);
    b = a ^ c;
    System.out.printf("A: %d\t B: %d\t C:%d\n",a,b,c);
    b++;
    System.out.printf("A: %d\t B: %d\t C:%d\n",a,b,c);
  }
}
```

Antes de descrever a saída e as operações utilizadas, vamos observar o comando **printf.**

Este comando permite uma saída formatada, onde cada **%d** receberá um valor inteiro, **\n** é utilizado para pular de linha e **\t** para representar uma tabulação. No caso, como existem três ocorrências de **%d** na expressão, ela receberá as três variáveis inteiras logo após a vírgula, preenchendo as lacunas **%d** com os valores dessas variáveis na ordem oferecida.

Observe agora a saída do programa.

A:5 B:32 C:7

A:5 B:5 C:7

A:5 B:2 C:7

A:5 B:3 C:7

Na primeira impressão temos os valores originais, declarados na inicialização, mas em seguida são feitas duas operações, onde **b** recebe seu valor decrementado do valor de **c**, sendo dividido por **a** em seguida.

A expressão **b /= a** é equivalente a **b = b / a**, e isto pode ser feito com qualquer operador quando a variável sofre uma operação sobre ela mesma. Igualmente, poderíamos ter escrito **b - = c** na primeira operação.

Fazendo as contas teremos os valores da segunda linha que foi impressa.

Em seguida temos uma operação binária entre **a** e **c**, com o valor armazenado em **b**. É uma operação de ou-exclusivo, onde o valor será 1 apenas se os dois bits comparados tiverem valores distintos.

A	0	1	0	1
C	0	1	1	1
a ^ c	**0**	**0**	**1**	**0**

Com esta operação justificamos na terceira impressão o valor de 2 para **b**, o qual é acrescido de 1 com o operador **++**, apresentando o valor 3 na quarta impressão.

Os operadores relacionais permitem a comparação entre valores, tendo como resultado um valor verdadeiro (**true**) ou falso (**false**), o que pode ser armazenado em uma variável booleana.

Podemos observá-los no quadro seguinte.

Operador	Operação
==	Compara a igualdade entre os termos
!=	Compara a diferença entre os termos
>	Compara se o primeiro é maior que o segundo
<	Compara se o primeiro é menor que o segundo
>=	Compara se o valor é maior ou igual
<=	Compara se o valor é menor ou igual

Estes operadores serão de grande importância para as estruturas de decisão e de repetição que utilizaremos em nossos programas, pois elas tratam diretamente com valores booleanos para o controle do fluxo de execução.

Nós também podemos combinar valores booleanos com o uso de operadores lógicos, os quais são expressos pela tabela-verdade a seguir:

A (Booleano 1)	B (Booleano 2)	A && B - AND	A || B - OR	! A - NOT
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

No exemplo seguinte podemos observar a utilização de operadores relacionais e lógicos.

```
package exemplo003;

public class Exemplo003 {
  public static void main(String[] args) {
    int a = 5, b = 32, c = 7;
    boolean x;
    x = a < b;
    System.out.println("PASSO 1: "+x);
    x = (b > a) && (c > b);
    System.out.println("PASSO 2: "+x);
    b /= a -= 1;
    c++;
    x = (b==c);
    System.out.println("PASSO 3: "+x);
  }
}
```

No primeiro passo temos **x** recebendo a comparação **a < b**, que é verdadeira.

No segundo passo, a comparação **b > a** é verdadeira, mas **c > b** é falsa, e com o operador lógico aplicado teremos o resultado final como falso.

Em seguida são feitas algumas operações aritméticas de forma concatenada, e que poderiam ser expressas como:

```
a = a – 1; // a passa a valer 4

b = b / a; // b é divido por a (4) e fica com 8

c = c + 1; // c passa a valer 8
```

Após estas operações, os valores de **b** e **c** são iguais, e a condição se torna verdadeira para o terceiro passo.

Podemos observar a saída resultante a seguir.

```
PASSO 1: true

PASSO 2: false

PASSO 3: true
```

Uma observação final é a de que na linguagem Java a divisão entre inteiros retorna um valor inteiro, enquanto para variáveis do tipo ponto flutuante será um valor real, ou seja, se efetuarmos a operação 5 / 2 teremos 2, enquanto 5.0 / 2 dará 2.5.

Estruturas de Decisão

Vídeo.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Após definidas as variáveis e os métodos de saída, o nosso próximo passo será a compreensão das estruturas de decisão existentes na sintaxe do Java.

O fluxo de execução sequencial indica que as diversas instruções serão executadas na ordem em que são apresentadas no código, mas existem formas de redirecionar o fluxo de execução para partes específicas.

É comum encontrarmos situações onde efetuamos determinadas ações apenas perante certas condições. Para isso, em termos de algoritmos, contamos com as estruturas de decisão.

Basicamente, contamos com duas estruturas de decisão:

- if..else
- switch..case

Estrutura if..else

A sintaxe da estrutura **if..else** é muito simples, conforme podemos observar a seguir.

```
if ( <<condição>> ) {

  // instruções para o caso verdadeiro

} else {

  // instruções para o caso falso

}
```

Se a **condição** especificada entre parênteses tiver valor verdadeiro, o primeiro bloco de comandos será executado, e se for falsa quem será executado é o segundo bloco.

O uso de chaves é necessário apenas quando temos blocos constituídos de mais de um comando, e o uso de **else** é opcional.

Vamos observar um pequeno exemplo de uso da estrutura **if..else**:

```
package exemplo004;
import java.util.Scanner;

public class Exemplo004 {
  public static void main(String[] args) {
    Scanner s1 =new Scanner(System.in);
    System.out.println("DIGITE UM NÚMERO:");
    int x = s1.nextInt();
    if(x%2==0)
      System.out.println("O NÚMERO É PAR");
    else
      System.out.println("O NÚMERO É ÍMPAR");
  }
}
```

Neste exemplo foi utilizado um objeto do tipo **Scanner**, inicializado com a entrada de dados padrão via teclado (**System.in**) para obter um número inteiro digitado pelo usuário. Isto pode ser observado na chamada **s1.nextInt( )**.

Após o usuário entrar com o número pelo teclado, ocorre o teste do if para saber se o resto da divisão por dois resulta em zero, imprimindo a mensagem “O NÚMERO É PAR” e, se o resto não for zero, imprimir “O NÚMERO É ÍMPAR” a partir do else.

Como os blocos tinham apenas uma instrução cada, o uso de chaves não foi necessário em nenhum dos dois.

Podemos observar uma possível saída para a execução desse código a seguir.

```
DIGITE UM NÚMERO:

5

O NÚMERO É ÍMPAR
```

Estrutura switch..case

Utilizamos a estrutura **switch..case** quando desejamos efetuar ações específicas para cada valor de uma variável. Um exemplo simples de utilização seria a construção de um menu de opções para um sistema.

A sintaxe básica utilizada por essa estrutura seria:

```
switch (<<variável>>) {
  case valor1:
    // instruções para o valor1
    break;
  case valor2:
    // instruções para o valor2
    break;
  default:
    // instruções para valores que não foram previstos
}
```

O comando **switch** irá desviar o fluxo de execução para os comandos **case**, de acordo com o valor da **variável**, sendo que o comando **default** será executado caso o valor não esteja entre aqueles que foram previstos.

Devemos observar que cada seção case deve ser terminada com o comando **break**, pois sua ausência irá causar a continuidade da execução, “invadindo” a seção seguinte.

Vamos observar um exemplo de utilização da estrutura **switch..case** a seguir:

```
package exemplo005;

import java.util.GregorianCalendar;
import java.util.Scanner;

public class Exemplo005 {
    public static void main(String[] args) {
       Scanner s1 =new Scanner(System.in);
       System.out.println("DIGITE O DIA ATUAL:");
       int d = s1.nextInt();
       System.out.println("DIGITE O MÊS ATUAL:");
       int m = s1.nextInt();
       System.out.println("DIGITE O ANO ATUAL:");
       int a = s1.nextInt();
       GregorianCalendar g1 = new GregorianCalendar(a,m-1,d);
       switch(g1.get(GregorianCalendar.DAY_OF_WEEK)){
           case 1:
              System.out.println("DOMINGO! FERIADO! :)");
              break;
           case 2:
           case 3:
           case 4:
           case 5:
           case 6:
              System.out.println("DIA ÚTIL! TRABALHANDO. :(");
              break;
           case 7:
              System.out.println("SÁBADO! FERIADO! :)");
              break;
           default :
              System.out.println("ALGO ESTÁ ERRADO....");
              break;
       }
    }
}
```

Aqui utilizamos, além do **Scanner**, um objeto do tipo **GregorianCalendar**. Esse objeto permitirá tratar elementos de data e hora, sendo inicializado com ano, mês (0 a 11) e dia.

Após a inicialização, utilizamos **get(GregorianCalendar.DAY_OF_WEEK)** para verificar o dia da semana, o qual é obtido como um valor inteiro de 1 a 7, correspondendo aos dias de domingo a sábado.

Em nosso exemplo, a estrutura **switch** utiliza esse valor para imprimir a frase equivalente a cada dia da semana, sendo a opção **default** executada apenas para valores fora da faixa de 1 a 7, algo que não ocorrerá devido ao uso de **GregorianCalendar**.

Podemos observar uma possível saída para a execução desse código a seguir.

```
DIGITE O DIA ATUAL:
17

DIGITE O MÊS ATUAL:
11

DIGITE UM NÚMERO:DIGITE O DIA ATUAL:DIGITE O ANO ATUAL:
2018

SÁBADO! FERIADO! :)
```

Estruturas de Repetição

Vídeo.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Outra forma de redirecionar o fluxo de execução é através do uso de estruturas de repetição. Elas servem para repetir a execução de um comando ou bloco de comandos enquanto determinada condição for verdadeira.

A linguagem Java permite a utilização das seguintes estruturas de repetição:

- for;
- while;
- do..while.

Estrutura for

A implementação de estruturas do tipo **PARA..FAÇA** utiliza a seguinte sintaxe:

for (<<inicialização>>; <<condição>>; **<<incremento>>**)

Inicialização

Definição dos valores iniciais das variáveis de controle.

Condição

Expressão booleana que determina a execução do comando ou bloco de comandos.

Incremento

Comandos para atualização das variáveis de controle.

Exemplo

Por exemplo, um loop de 1 a 5 seria **for(int i=1 ; i<=5 ; i++)**.

Estruturas deste tipo são utilizadas quando trabalhamos com uma regra de execução com início e fim bem determinados, como no caso de passeios em vetores, ou quando somamos os valores de uma sequência. Vamos observar o exemplo a seguir.

```
public class Exemplo006 {
 public static void main(String[] args) {
   // Calculo do valor médio da sequencia y = f(x) = x * x
  // Media = Somatorio dos valores / quantidade
  // Limites 1 a 5
  double soma = 0.0;
  for(int x=1; x<=5; x++)
    soma += Math.pow(x, 2);
    // eleva x a potência 2 e acumula
  System.out.println(soma/5);
 }
}
```

Neste exemplo temos o cálculo do valor médio de uma sequência entre os limites de 1 a 5, considerando a função $\mathbf{y = x^2}$. Foi utilizado o método **Math.pow**, que eleva um número a uma potência qualquer, para calcular o quadrado de x.

Ao executar o programa teremos a impressão do valor **11**, o que corresponde exatamente ao valor médio, dado por (1 + 4 + 9 + 16 + 25) / 5.

Estruturas while e do..while

A implementação de estruturas do tipo **ENQUANTO..FAÇA** utiliza a seguinte sintaxe:

```
while ( << condição[1] >> ) {

  // Bloco de Comandos

}
```

Com relação às estruturas do tipo **FAÇA..ENQUANTO**, a sintaxe utilizada seria:

```
Do {

  // Bloco de Comandos

} while ( << condição[2] >> );
```

No exemplo seguinte podemos observar a utilização do comando **while**.

```
public class Exemplo007 {
 public static void main(String[] args) {
   int[] x1 = {21, 32, 15, 27, 33, 17};
   int posicao = 0;
   int soma = 0;
   while(posicao< x1.length){
     // Enquanto for menor que o tamanho do vetor
     soma += x1[posicao];
     posicao++;
   }
   System.out.println(soma);
 }
}
```

Neste exemplo é criado um vetor inicializado na criação, assumindo 6 posições, ou seja, terá os índices de 0 a 5.

x1[0] = 21	x1[0] = 21	x1[2] = 15	x1[3] = 27	x1[4] = 33	x1[5] = 17

Em seguida procedemos ao somatório desses valores com o uso de um comando **while**, tendo como condição **posicao< x1.length**. Como o length neste caso é 6, a posição varia de 0 a 5.

A cada rodada, o valor do vetor na posição equivalente é adicionado à variável soma, e não podemos esquecer de incrementar a posição, pois caso contrário teríamos uma repetição infinita.

O uso de **length** permite que seja alterada a quantidade de elementos de **x1** na inicialização sem a necessidade de modificações no restante do código.

Ao final teremos impresso o valor 145, que corresponde à soma 21+32+15+27+33+17.

O comando **do..while** tem funcionamento similar, mas o teste é feito ao final, o que garante pelo menos uma execução.

Podemos observar um exemplo de utilização do comando **do..while** a seguir.

```
public class Exemplo008{

import java.util.Scanner;

public class Exemplo008 {
 public static void main(String[] args) {
   Scanner teclado = new Scanner(System.in);
   int soma = 0;
   do{
     System.out.print("Digite um numero (0 para sair):");
     valor = teclado.nextInt();
     soma += valor;
   } while(valor!=0);
   System.out.printf("\nA soma dos numeros "+
                     "digitados e: %d\n",soma);
 }
}
```

Neste nosso exemplo serão solicitados os números ao usuário até que ele digite 0, sendo apresentado a seguir o somatório dos números digitados por ele.

Podemos verificar uma possível saída desse programa a seguir.

```
Digite um numero (0 para sair):13
Digite um numero (0 para sair):32
Digite um numero (0 para sair):21
Digite um numero (0 para sair):7
Digite um numero (0 para sair):0

A soma dos numeros digitados e: 73
```

Atividade

1. Uma das características do Java é a possibilidade de execução em plataformas distintas sem a necessidade de recompilar o código fonte. Qual componente permite isso?

a) Garbage Collector
b) JVM
c) JDK
d) NetBeans
e) Eclipse

2. Observando a sintaxe do comando for, e considerando que todas as variáveis utilizadas foram previamente declaradas, qual das opções seguintes irá necessariamente gerar um erro de compilação?

a) for(a=1,b=20; a < b; a++,b+=2)
b) for(; a++ < 10;)
c) for(a=1; a=10; a++)
d) for(; ;)
e) for(a=200; a>1; a/=2)

3. Implemente um programa em Java para imprimir os 10 primeiros números primos que ocorrem após o número 1. Lembrando que um número primo é aquele divisível apenas por ele mesmo e por 1.

Notas

Condição (while)[1]

Expressão booleana que determina a execução do comando ou bloco de comandos.

Condição (do...while)[2]

Expressão booleana que determina a continuidade da execução do bloco de comandos.

Referências

CORNELL, G.; HORSTMANN, C. **Core java**. 8. São Paulo: Pearson, 2010.

DEITEL, P.; DEITEL, H. **Java, como programar**. 8. São Paulo: Pearson, 2010.

FONSECA, E. **Desenvolvimento de software**. Rio de Janeiro: Estácio, 2015.

SANTOS, F. **Programação I**. Rio de Janeiro: Estácio, 2017.

Próxima aula

- Princípios da orientação a objetos;
- Palavras da sintaxe Java para Orientação a Objetos;
- Elementos concretos e abstratos.

Explore mais

Leia os textos:

- Tutorial para início rápido do Java do NetBeans IDE. <https://netbeans.org/kb/docs/java/quickstart_pt_BR.html>
- Fundamentos da linguagem Java. <https://www.ibm.com/developerworks/br/java/tutorials/j-introtojava1/index.html>

Assista ao vídeo:

- Instalação do NetBeans e JDK. <https://www.youtube.com/watch?v=GNUGNfIIGhA>