

Java para Concursos – Módulo 1

Vitor Almeida

<http://www.itnerante.com.br/profile/vitor>

Agenda

- 1 Bibliografia
- 2 Tópicos Iniciais
- 3 Sintaxe Básica
- 4 Objetos e Classes

1 Bibliografia

- Documentação oficial:
<http://www.oracle.com/technetwork/java/javase/documentation/index.html>
- Deitel. Java como programar. 8ª ed. 2010.
- Apostilas da Caelum:
<http://www.caelum.com.br>
- JavaFree.org:
<http://javafree.uol.com.br/>

2 Tópicos Iniciais

2.1 Características de Java

- Java refere-se tanto a uma linguagem de programação quanto a uma plataforma.
- É uma linguagem de alto nível com as seguintes características: simples, orientada a objetos, distribuída, multithread, dinâmica, de arquitetura neutra, portátil, de alta performance, robusta e segura.

2.1 Características de Java

- A Tecnologia Java divide-se em:
 - Java Standard Edition: desenvolvimento desktop.
 - Java Enterprise Edition: Java para Web.
 - Java Micro Edition: desenvolvimento de aplicações para celulares, palmtops etc.
 - Java Embedded: Plataforma para desenvolvimento de programas para dispositivos diversos.
 - JavaFX: Desenvolvimento de Rich Internet Applications (RIA).

2.2 Processo de desenvolvimento de software

- São cinco fases:
 - Edição
 - Compilação
 - Carregamento
 - Verificação
 - Execução

2.2 Processo de desenvolvimento de software

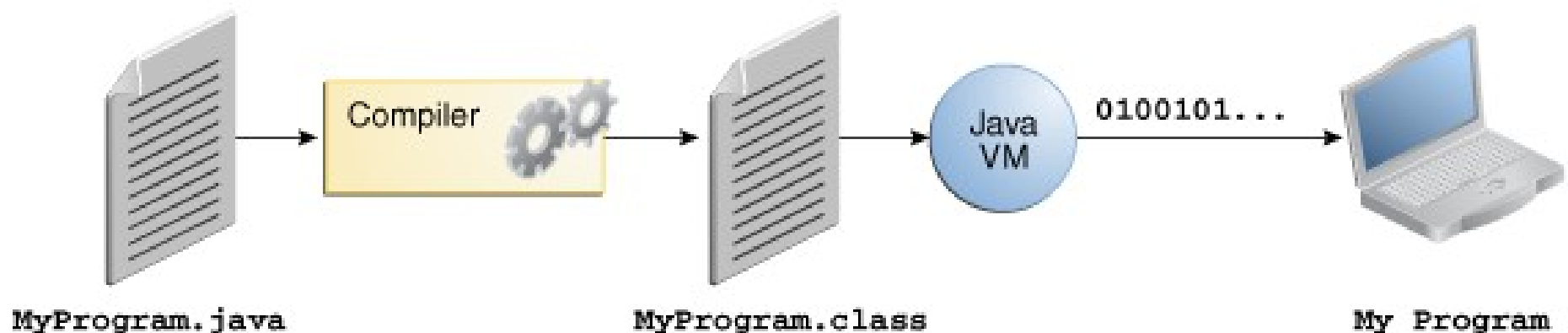
- Edição:
 - O programa é criado em um editor e armazenado em disco em um arquivo .java.
- Compilação:
 - O compilador gera bytecodes a partir do código-fonte e os armazena em disco em um arquivo .class.

2.2 Processo de desenvolvimento de software

- Carregamento:
 - O carregador de classe lê os arquivos .class que contêm bytecodes a partir do disco e coloca estes bytecodes na memória.
- Verificação:
 - O verificador de bytecode confirma que todos os bytecodes são válidos e não violam restrições de segurança do Java.

2.2 Processo de desenvolvimento de software

- Execução:
 - Para executar o programa, a JVM lê os bytecodes e os compila (isto é, traduz) no momento certo (ou Just-In-Time - JIT) para uma linguagem que o computador consegue entender.

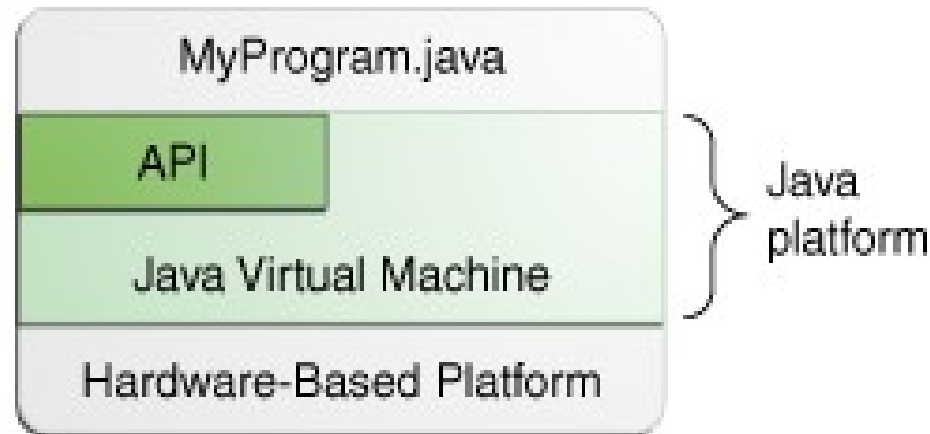


2.3 A plataforma Java

- A plataforma Java é uma camada de software que roda em cima da plataforma de hardware.
- Contem dois componentes:
 - Java Virtual Machine
 - Java Application Programming Interface (API)

2.3 A plataforma Java

- Esquemáticamente, tem-se:



2.4 JDK

- Para escrever um programa é preciso um Java Development Kit - JDK. As duas ferramentas do JDK mais famosas são:
 - javac.exe
 - java.exe

2.5 JRE

- Java Runtime Environment - JRE é o software necessário para rodar programas Java.
- É composta por:
 - JVM
 - Biblioteca de classes
 - Arquivos de suporte

Questão de concurso

1 Cespe TRE-RJ 2012 A linguagem de programação Java é muito utilizada por ter como característica gerar um código independente de plataforma que pode ser executado em qualquer arquitetura e sistema operacional que tenha o sistema Java.

Questão de concurso

1 Cespe TRE-RJ 2012 A linguagem de programação Java é muito utilizada por ter como característica gerar um código independente de plataforma que pode ser executado em qualquer arquitetura e sistema operacional que tenha o sistema Java.

Questão de concurso

2 Cespe TRT-RN 2011 A linguagem de programação Java, em razão de sua portabilidade — uma vez que o compilador Java converte o código fonte em bytecodes, executados por uma máquina virtual — é bastante utilizada para oferecer conteúdos dinâmicos na Web.

Questão de concurso

2 Cespe TRT-RN 2011 A linguagem de programação Java, em razão de sua portabilidade — uma vez que o compilador Java converte o código fonte em bytecodes, executados por uma máquina virtual — é bastante utilizada para oferecer conteúdos dinâmicos na Web.

Questão de concurso

3 Cesgranrio Transpetro 2011 Muito utilizada para desenvolvimento de aplicativos Web, a tecnologia Java tem como principal característica gerar aplicações que rodam em qualquer dispositivo que tenha acesso a Internet, utilizando, entre outros recursos, o software

- a) JBC (Java Bytecode Console)
- b) JDB (Java Developer Builder)
- c) MS (Java Management Server)
- d) JAC (Java Application Controler)
- e) JVM (Java Virtual Machine)

Questão de concurso

3 Cesgranrio Transpetro 2011 Muito utilizada para desenvolvimento de aplicativos Web, a tecnologia Java tem como principal característica gerar aplicações que rodam em qualquer dispositivo que tenha acesso a Internet, utilizando, entre outros recursos, o software

- a) JBC (Java Bytecode Console)
- b) JDB (Java Developer Builder)
- c) MS (Java Management Server)
- d) JAC (Java Application Controler)
- e) JVM (Java Virtual Machine)

Questão de concurso

4 Cespe BASA 2012 No Java, a JRE possui tudo que é necessário para desenvolver programas em Java.

Questão de concurso

4 Cespe BASA 2012 No Java, a JRE possui tudo que é necessário para desenvolver programas em Java.

Gabarito

- 1 C
- 2 C
- 3 E
- 4 E

3 Sintaxe Básica

3.1 Tipos primitivos

- Tipos primitivos que representam números inteiros:
 - **byte** (8 bits): -128 (-2^7) até 127 ($2^7 - 1$)
 - **short** (16 bits): -32.768 (-2^{15}) até 32.767 ($2^{15} - 1$)
 - **int** (32 bits): -2^{31} até $2^{31} - 1$
 - **long** (64 bits): -2^{63} até $2^{63} - 1$

3.1 Tipos primitivos

- Tipos primitivos que representam números decimais:
 - `float` (32 bits)
 - `double` (64 bits)
- `float` reserva 23 bits para os números significativos e 9 bits para o expoente.
- `double` reserva 52 bits e 12 bits, respectivamente.
- Exemplificando, 1,23 pode ser representado como 123×10^{-2} :
 - 123 é o número significativo e -2 é o expoente.

3.1 Tipos primitivos

- Outros tipos primitivos:
 - **char** (16 bits): Armazena um caractere Unicode
 - **boolean** (1 bit): Pode conter um de dois possíveis estados: true ou false

3.2 Variáveis

- As variáveis são locais onde os dados são armazenados durante a execução do programa.
- Cada variável tem seu espaço individual de memória.
- Java é fortemente tipada, o que significa que cada variável deve ter um tipo declarado.

3.2 Variáveis

- Além do tipo, cada variável deve ter um nome ou identificador.
- As regras para criação de identificadores:
 - Só pode conter letras, _, \$ ou dígitos [0-9]
 - Precisa iniciar por uma letra, _ ou \$
 - Não pode ser palavra-chave, “false”, “true” ou “null”
 - Precisa ser único em determinado escopo

3.2 Variáveis

- Palavras-chave (keywords):

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

3.2 Variáveis

- Java é *case sensitive*.
- Exemplos de nomes válidos:
 - CpfValido (que é diferente de cpfValido)
 - \$tremruim
 - _mínimo
- Exemplos de nomes inválidos:
 - 2x
 - sangue-azul

3.2 Variáveis

- Na declaração da variável, escreve-se o tipo, o identificador da variável e, opcionalmente, o valor a ser armazenado:

```
1 byte x;  
2 int linhas;  
3 char c;  
  
4 int a,b;  
  
5 byte x = 12;  
6 int linhas = 1000;  
7 char c = 'x';
```


3.2 Variáveis

- Caso não seja inicializada, a variável recebe o valor padrão:
 - boolean: false
 - byte, short, int, long, float, double: 0
 - char: \u0000

3.3 Constantes e literais

- Constantes são variáveis cujos valores uma vez definidos não podem ser mudados.
- A palavra-chave **final** é utilizada na declaração:

```
1 final int LINHAS = 50;  
2 final boolean PERMITIR_ACESSO = true;
```

3.3 Constantes e literais

- Literal nada mais é do que a representação de um valor no código. Existem três tipos de literal: tipos primitivos, null e string.
- Veja exemplos de literal abaixo:

```
1 int x = 50;  
2 boolean teste = false;
```

3.3 Constantes e literais

- Escrita de números inteiros:
 - Números decimais: 123
 - Hexadecimais: 0XA12
 - Base 8: 0123
- Sempre observe o limite da capacidade de memória da variável. O código abaixo daria um erro de compilação:

```
1 byte x = 250;
```

3.3 Constantes e literais

- Literais do tipo long precisam do sufixo L ou l:

```
1 long a = 9876543210L;
```

- long, int, byte e short também podem ser representados em binário com o prefixo 0B ou 0b:

```
2 byte twelve = 0B1100;
```

3.3 Constantes e literais

- É possível inserir `_` para facilitar a leitura de números muito grandes:

```
1 int million = 1000000;  
2 int million2 = 1_000_000;
```

- Literais de ponto flutuante (float e double) possuem quatro partes: número inteiro, ponto decimal, parte decimal e um expoente opcional.

3.3 Constantes e literais

- Você pode representar literais do tipo float de três formas:
 - dígitos . [dígitos] [parte exponencial] f ou F
 - . dígitos [parte exponencial] f ou F
 - dígitos [parte exponencial] f ou F
- Literais do tipo double é só trocar o f ou F por d ou D.

3.3 Constantes e literais

- A parte exponencial é representada da seguinte forma:
 - Indicador [sinal]inteiro.
- O indicador pode ser e ou E.
- O sinal pode ser - ou +, sendo opcional no caso positivo.
- O inteiro é um número inteiro.

3.3 Constantes e literais

- Exemplos:
 - 2e-1f
 - 8.f
 - .5F
 - 0d
 - 3.14d
 - 9.0001e+12D

3.3 Constantes e literais

- Literais do tipo boolean podem ser true ou false.
- Literais do tipo char são caracteres unicode ou sequências de escape envolvidos por ' '.
- Exemplos:
 - 'Z'
 - '0'
 - 'ü'
 - '%'

3.3 Constantes e literais

- Sequências de escape são iniciadas por \ e representam caracteres unicode que não podem ser escritos com o teclado ou representam funções especiais.
- Exemplos:
 - '\b' backspace
 - '\t' tab
 - '\\' barra invertida
 - '\u2299' ⊗

3.4 Conversão de tipos primitivos

- Decorre da necessidade de atualização de valores de variáveis que estão armazenados em variáveis de tipo diferente.
- Quando o tipo é o mesmo, não tem problema:

```
1 int a = 90;  
2 int b = a;
```

3.4 Conversão de tipos primitivos

- Existem dois tipos de conversão. Na conversão implícita, a variável que receberá o valor de outra deverá ter um tamanho igual ou maior:
 - Atribuir a uma variável long o valor armazenado em um int
 - Atribuir a uma variável int o valor armazenado em um char
 - Atribuir a uma variável double o valor armazenado em um float

3.4 Conversão de tipos primitivos

- A conversão é implícita porque não existe uma sintaxe especial:

```
1 int a = 10;  
2 long b = a;
```

- A única possibilidade de perda de informação ocorre na conversão de int ou long para float (ou long para double).

3.4 Conversão de tipos primitivos

- Na conversão explícita, uma variável com capacidade menor recebe o valor armazenado em uma variável com tipo de capacidade maior.
- É preciso informar o tipo desejado após a conversão:

```
1 long a = 10;  
2 int b = (int) a;
```

3.4 Conversão de tipos primitivos

- A conversão explícita pode acarretar perda de informação. Na conversão abaixo b passa a armazenar 1286608618 (!!!).

```
1 long a = 9876543210L;  
2 int b = (int) a;
```

- Conversão explícita introduz erros no programa e deve ser evitada.

3.5 Operadores

- Os operadores realizam operações entre operandos.
- Existem muitos tipos de operadores, como adição, subtração, divisão etc.
- Existem operadores unários, binários e ternários em Java.
- Exemplo:

`1 x + 4;`

3.5 Operadores

- Operadores unários:
 - Operador menos (-): retorna o negativo de um operando.
 - Operador mais (+): retorna o valor do operando.

```
1 float x = 4.5f;  
2 float y = +x; // y = 4.5f  
3 float z = -x; // z = -4.5f
```

3.5 Operadores

- Operadores unários:
 - Operador de incremento (++): soma 1 ao valor do operando.
 - Ele pode ser utilizado antes (prefixado) ou depois do operando (pós-fixado).
 - Quando o valor de uma expressão com o ++ é atribuído a uma variável, no prefixado, soma-se 1 ao operando e se atribui o valor. No pós-fixado, se atribui o valor e depois soma um.

3.5 Operadores

- Operadores unários:
 - Operador de incremento (++):

```
1 int x = 4;    // x = 4
2 int y = ++x;  // y = 5, x = 5
3 int z = x++;  // z = 5, x = 6
```

3.5 Operadores

- Operadores unários:
 - Operador de decremento (--): faz exatamente o contrário do ++, diminui 1 do operando.

```
1 int x = 4;    // x = 4
2 int y = --x;  // y = 3, x = 3
3 int z = x--;  // z = 3, x = 2
```

3.5 Operadores

- Operadores unários:
 - Operador lógico de negação (!): utilizado em variáveis, literais e expressões do tipo boolean.
 - Inverte o valor armazenado (true vira false e false vira true).

```
1 boolean x = false;  
2 boolean y = !x;           //y = true
```

3.5 Operadores

- Operadores unários:
 - Operador not (~): o resultado é o complemento bit a bit do operando.

```
1 int x = 2;  
2 int y = ~x; // y = -3
```

```
  2 = 0000 0000 0000 0000 0000 0000 0000 0010  
-3 = 1111 1111 1111 1111 1111 1111 1111 1101
```

3.5 Operadores

- Operadores aritméticos:
 - Soma (+), subtração (-), multiplicação (*) e divisão (/).
 - Módulo (%): retorna o resto da divisão do primeiro operando pelo segundo.

```
1 int x = 2;  
2 int y = x * 2 + 5; // y = 9  
3 int z = y % x;    // z = 1
```


3.5 Operadores

- Operadores de igualdade:
 - Operador de igualdade (==): retorna true se os operandos forem iguais e false se forem diferentes.
 - Operador de desigualdade (!=): retorna false se os operandos forem iguais e true se forem diferentes.

```
1 boolean x = true;  
2 boolean y = true;  
3 boolean z = x != y;    //z = false  
4 boolean a = x == y;    //a = true
```

3.5 Operadores

- Operadores relacionais: os operadores relacionais realizam comparações e retornam true ou false de acordo com os valores dos operandos.
- São cinco: >, >=, <, <= e instanceof.

```
1 int x = 5;  
2 int y = 2;  
3 boolean z = x > y; //z = true  
4 boolean a = x <= y; //a = false
```

3.5 Operadores

- Operadores condicionais:
 - Operador e (&&): retorna true apenas se ambos os operandos forem true.
 - Operador ou (||): retorna false apenas se ambos os operandos forem false.

```
1 int x = 5;  
2 int y = 2;  
3 boolean z = (x > y) && (x != y); //z = true  
4 boolean a = (x <= y) || (x < y); //a = false
```

3.5 Operadores

- Operadores condicionais:
 - Operador ?: é o único ternário. Avalia o primeiro operando, retorna o segundo operando caso o primeiro resulte em true ou retorna o terceiro caso seja false.

```
1 int x = 2;  
2 int y = 5;  
3 int z = (y == x) ? 1 : 2 // z = 2
```

3.5 Operadores

- Operadores de deslocamento: Os operadores de bit shift (deslocamento de bits) são usados para deslocar uma certa quantidade de bits de um valor inteiro.
 - Operador de deslocamento à esquerda (<<): desloca bits para a esquerda e completa os espaços deixados na direita com 0.
 - $A \ll B = A * 2^B$ ($1 \ll 3 = 8$)
 - Operador de deslocamento à direita (>>): desloca bits para a direita e completa os espaços deixados na esquerda com 0.
 - $A \gg B = A / 2^B$ ($16 \gg 1 = 8$)

3.5 Operadores

- Operadores de deslocamento:
 - Operador de deslocamento lógico à direita (>>>): desloca bits para a direita e completa os espaços deixados na esquerda com o bit de sinal (bit mais significativo), ou seja, se o bit de sinal for 1 preenche os espaços com 1, caso contrário preenche os espaços com 0.

3.5 Operadores

- Operadores de deslocamento:
 - Na prática:

```
1 int b = -1431655766;  
2 // 101010101010101010101010101010101010101010101010  
3 b << 3;  
4 // 101010101010101010101010101010101010101010101010000  
5 b >> 3  
6 // 0001010101010101010101010101010101010101010101010  
7 b >>> 3  
8 // 11110101010101010101010101010101010101010101010
```

3.5 Operadores

- Operadores de atribuição:
 - São 12: =, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^= e |=.
 - Exceto o =, consistem de dois operadores aglutinados:
 - $x -= 5$ é o mesmo que $x = x - 5$

```
1 int x = 2;  
2 x *= 5; // x = 10
```


3.5 Operadores

- Operadores bitwise (& | ^):
 - Realizam uma operação bit a bit com dois operandos do tipo int.
 - O & corresponde ao e, o | ao ou e o ^ ao ou exclusivo:
 - $0xFFFF \& 0x0000 = 0x0000$
 - $0xF0F0 \& 0xFFFF = 0xF0F0$
 - $0xFFFF | 0x000F = 0xFFFF$
 - $0xFFFF0 \wedge 0x00FF = 0xFF0F$

3.5 Operadores

- Operadores lógicos (& | ^):
 - Realizam uma operação lógica com dois boolean.
 - O & corresponde ao e, o | ao ou e o ^ ao ou exclusivo:
 - `true & true = true`
 - `true & false = false`
 - `true | false = true`
 - `false | false = false`
 - `true ^ true = false`
 - `false ^ false = false`

3.5 Operadores

- Precedência de operadores:
 - Alguns operadores são executados primeiro em uma expressão, independente da ordem em que aparecem.

```
1 int a = 1;
2 int b = 2;
3 int c = 3;
4 int d = a + b * c;    //d = 7
5 int d = (a + b) * c;  //d = 9
```

3.5 Operadores

- Precedência de operadores
 - Lista de operadores de acordo com a precedência:
 - Operadores unários, criação ou cast, `*/`, `%`, `+-`, `<<` `>>` `>>>`, `<` `>` `>=` `<=`, `==` `!=`, `&`, `^`, `|`, `&&`, `||`, `?:`, operadores de atribuição.

3.5 Operadores

- Promoção:
 - Alguns operadores causam uma promoção automática do tipo da literal (exemplo: byte para int).
 - Por exemplo, o código abaixo resulta em erro porque o operador - retorna um inteiro

```
1 byte a = 1;  
2 byte b = -a;
```

3.5 Operadores

- Promoção
 - Operadores unários em operandos do tipo byte, char ou short resultam em int.
 - Operadores binários:
 - Se byte ou short, vira int; se float, vira float; se double, vira double; se long, vira long.
 - A solução é usar cast.

```
1 short x, y = 200;  
2 short z = (short) (x + y);
```

3.6 Comentários

- É uma boa prática comentar o código, explicando funcionalidades e situações específicas.
- Existem dois tipos:
 - Comentários tradicionais: `/* */`
 - Comentários de uma linha: `//`

3.6 Instruções

- Um programa é feito de instruções.
- Algumas instruções, como o if, o while e o for determinam o fluxo de execução do programa.
- Enquanto que expressões são conjuntos de operandos e operadores que serão avaliados, uma instrução resulta em uma ação que será realizada pelo computador

3.6 Instruções

- Uma instrução é terminada por um ;
- Instruções estão organizadas em blocos (pedaços de códigos envolvidos por { }).
- Um bloco possui instruções, classes locais e variáveis locais.

```
1 {  
2 short x, y = 200;  
3 short z = (short) (x + y);  
4 }
```

3.7 Instrução if

- É uma instrução de decisão. Pode vir em duas formas:

```
1 if (expressão booleana) {  
2 //instrução(ões)  
3 }
```

```
4 if (expressão booleana) {  
5 //instrução(ões)  
6 }  
7 else {  
8 //instrução(ões)  
9 }
```

3.7 Instrução if

- A seleção múltipla é feita com uma série de instruções else

```
1 if (expressão booleana 1) {  
2 //instrução(ões)  
3 } else if (expressão booleana 2) {  
4 //instrução(ões)  
5 }  
...  
6 else {  
7 //instrução(ões)  
8 }
```

3.7 Instrução if

- Exemplo 1

```
1 if (a > 2) {  
2   System.out.println("a > 2");  
3 } else {  
4   System.out.println("a < 2");  
5 }
```

3.7 Instrução if

- Exemplo 2

```
1 if (a == 1) {  
2 System.out.println("one");  
3 } else if (a == 2) {  
4 System.out.println("two");  
5 } else if (a == 3) {  
6 System.out.println("three");  
7 } else {  
8 System.out.println("invalid");  
9 }
```

3.8 Instrução while

- Muitas vezes é preciso repetir um bloco de código durante a execução do programa.
- Exemplo, suponha que você quer um programa que dê três bips com um intervalo de meio segundo entre eles.
- Para dar um beep, o comando abaixo resolve:

```
1 java.awt.Toolkit.getDefaultToolkit().beep();
```

3.8 Instrução while

- Mas pra dar o tempo de meio segundo, o código é esse:

```
1 try {  
2 Thread.currentThread().sleep(500);  
3 } catch (Exception e) {  
4 }
```

- Aí resolveríamos o problema assim:

3.8 Instrução while

```
1 java.awt.Toolkit.getDefaultToolkit().beep();  
2 try {  
3     Thread.currentThread().sleep(500);  
4 } catch (Exception e) {  
5 }  
6 java.awt.Toolkit.getDefaultToolkit().beep();  
7 try {  
8     Thread.currentThread().sleep(500);  
9 } catch (Exception e) {  
0 }  
1 java.awt.Toolkit.getDefaultToolkit().beep();
```


3.8 Instrução while

- Esta não é uma opção inteligente!
 - Por que você pode não saber a priori a quantidade de vezes que se deve repetir o pedaço de código.
 - Porque se você precisar modificar algum código do bloco você precisará fazer a mudança em todas as vezes em que o código se repetir.
- A solução elegante é usar uma estrutura de repetição, como o while.

3.8 Instrução while

- A sintaxe da instrução while é a seguinte:

```
1 while (expressão booleana) {  
2 //instrução(ões)  
3 }
```

3.8 Instrução while

- Exemplo 1:

```
1 int i = 0;
2 while (i < 3) {
3     System.out.println(i);
4     i++;
5 }
```

3.8 Instrução while

- Exemplo 2:

```
1 int j = 0;
2 while (j < 3) {
3     java.awt.Toolkit.getDefaultToolkit().beep();
4     try {
5         Thread.currentThread().sleep(500);
6     } catch (Exception e) {
7     }
8     j++;
9 }
```

3.8 Instrução do-while

- O do-while é uma estrutura de repetição bem parecida com o while. Sua sintaxe é:

```
1 do {  
2 //instrução(ões)  
3 } while (expressão booleana);
```

3.8 Instrução do-while

- Exemplo 1:

```
1 int i = 0;
2 do {
3     System.out.println(i);
4     i++;
5 } while (i < 3);
6 //Resultado no console: 0 1 2
```

3.8 Instrução do-while

- Exemplo 2:

```
1 int j = 4;  
2 do {  
3     System.out.println(j);  
4     j++;  
5 } while (j < 3);  
6 //Resultado no console: 4
```

3.9 Instrução for

- O for também é uma estrutura de repetição, porém, é mais complexo.
- O for começa com uma inicialização, a seguir uma expressão booleana é avaliada.
- Caso seja false, o for termina.
- Caso seja true, um bloco de instruções é executado, bem como uma instrução de atualização.

3.9 Instrução for

- O for tem a seguinte sintaxe:

```
1 for (init, expressão booleana, update) {  
2 //instrução(ões)  
3 }
```

3.9 Instrução for

- Exemplo 1:

```
1 for (int i = 0; i < 3; i++) {  
2   System.out.println(i);  
3 }  
4 //Resultado no console: 0 1 2
```

3.9 Instrução for

- Exemplo 2:

```
1 for (int i = 0; i < 3; i++) {  
2 if (i % 2 == 0) {  
3 System.out.println(i);  
4 }  
5 }  
6 //Resultado no console: 0 2
```

3.9 Instrução for

- Exemplos 3 e 4:

```
1 for (int i = 0; i < 3; i += 2) {  
2 System.out.println(i);  
3 }  
4 //Resultado no console: 0 2
```

```
1 for (int i = 3; i > 0; i--) {  
2 System.out.println(i);  
3 }  
4 //Resultado no console: 3 2 1
```

3.9 Instrução for

- Exemplos 5 e 6:

```
1 int j = 0;
2 for ( ; j < 3; j++) {
3     System.out.println(j);
4 }
```

```
1 int k = 0;
2 for ( ; k < 3; ) {
3     System.out.println(k);
4     k++;
5 }
```

3.9 Instrução for

- Estruturas de repetição aninhadas

```
1 for (int i = 0; i < 3; i++) {  
2   for (int j = 0; j < 3; j++) {  
3     System.out.println(i, j);  
4   }  
5 }  
6 //Resultado no console: 00 01 02 10 11 12  
7 //20 21 22
```

3.9 Instrução for

- O for e o while são equivalentes:

```
1 //inicialização do contador
2 while (expressão booleana) {
3 //instrução(ões)
4 //update no contador
5 }
```

```
1 for (init, expressão booleana, update) {
2 //instrução(ões)
3 }
```

3.10 Instrução break

- O break serve para quebrar o fluxo de execução de um for, while, do-while ou switch, saindo do bloco de instruções.
- Exemplo 1:

```
1 int i = 0;
2 while (true) {
3     System.out.println(i);
4     i++;
5     if (i > 2) {
6         break;
7     } //Resultado no console: 0 1 2
```


3.10 Instrução break

- Exemplo 2:

```
1 int m = 0;
2 for ( ; ; ) {
3     System.out.println(m) ;
4     m++;
5     if (m > 4) {
6         break;
7     }
8 }
9 //Resultado no console: 0 1 2 3 4
```

3.10 Instrução break

- O break pode ser usado com um label

```
1 fora: for (int i = 0; i < 3; i++) {  
2   for (int j = 0; j < 3; j++) {  
3     if (j == 1) {  
4       break fora;  
5     }  
6   }  
7 } //ao término i = 0 e j = 1
```

3.11 Instrução continue

- O continue é parecido com o break, mas quebra apenas a iteração atual e o fluxo de execução continua na próxima iteração.
- O código abaixo imprime os números de 1 a 9, exceto 5.

```
1 for (int i = 0; i < 10; i++) {  
2   if (i == 5) {  
3     continue;  
4   }  
5   System.out.println(i);  
6 }
```

3.12 Instrução switch

- O switch é uma alternativa ao uso de múltiplos else if.
- O switch permite que determinado bloco de código seja executado a partir do resultado de uma expressão que pode ser um int, uma String ou uma enumeração.
- A seguir, a sintaxe do switch.

3.12 Instrução switch

```
01 switch(expressão) {  
02 case valor_1:  
03 //instrução(ões);  
04 break;  
05 case valor_2:  
06 //instrução(ões);  
07 break;  
...  
08 case valor_n:  
09 //instrução(ões);  
10 break;  
11 default:  
12 //instrução(ões);  
13 }
```

3.12 Instrução switch

```
01 int i = ...;
02 switch (i) {
03     case 1 :
04         System.out.println("Um jogador");
05         break;
06     case 2 :
07         System.out.println("Dois jogadores");
08         break;
09     case 3 :
10         System.out.println("três jogadores");
11         break;
12     default:
13         System.out.println("Número inválido");
14 }
```

Questão de concurso

1 ESAF CGU 2012. Os tipos primitivos da linguagem Java são

- a) boolean, byte, narrow, int, wide, fixed, double, char.
- b) boolean, byte, short, int, long, float, double, char.
- c) buffered, byte, double-byte, single, long, float, double, char.
- d) logical, boolean, short, local, extended, float, double, cast.
- e) boolean, byte, short, integral, partial, long, float, char.

Questão de concurso

1 ESAF CGU 2012. Os tipos primitivos da linguagem Java são

- a) boolean, byte, narrow, int, wide, fixed, double, char.
- b) boolean, byte, short, int, long, float, double, char.**
- c) buffered, byte, double-byte, single, long, float, double, char.
- d) logical, boolean, short, local, extended, float, double, cast.
- e) boolean, byte, short, integral, partial, long, float, char.

Questão de concurso

2 FCC INFRAERO 2011 No Java, um tipo inteiro (int) utiliza quatro bytes para armazenamento. A faixa máxima possível de valores inteiros para se armazenar em uma variável do tipo primitivo int é de:

- a) -8388608 a 8388607.
- b) -128 a 127.
- c) -32768 a 32767.
- d) -9223372036854775808 a 9223372036854775807.
- e) -2147483648 a 2147483647.

Questão de concurso

2 FCC INFRAERO 2011 No Java, um tipo inteiro (int) utiliza quatro bytes para armazenamento. A faixa máxima possível de valores inteiros para se armazenar em uma variável do tipo primitivo int é de:

- a) -8388608 a 8388607.
- b) -128 a 127.
- c) -32768 a 32767.
- d) -9223372036854775808 a 9223372036854775807.
- e) -2147483648 a 2147483647.

Questão de concurso

3 FCC TJ-AL 2008 NÃO são nomes válidos em Java:

- a) _Real e \$real
- b) um1 e dois2
- c) 3tres e tres3
- d)Codigo e codigo
- e) cod_valor e cod\$valor

Questão de concurso

3 FCC TJ-AL 2008 NÃO são nomes válidos em Java:

- a) _Real e \$real
- b) um1 e dois2
- c) 3tres e tres3
- d)Codigo e codigo
- e) cod_valor e cod\$valor

Questão de concurso

4 Cespe MEC 2011 Int, byte, double e char são alguns tipos primitivos de variáveis suportadas pelo Java. As linhas abaixo declaram, corretamente, var1, var2 e var3 como sendo do tipo int.

```
int var1;
```

```
int var2;
```

```
int var3;
```

Outra forma também correta para a mesma declaração seria a que se segue

```
int var1, var2, var3;
```

Questão de concurso

4 Cespe MEC 2011 Int, byte, double e char são alguns tipos primitivos de variáveis suportadas pelo Java. As linhas abaixo declaram, corretamente, var1, var2 e var3 como sendo do tipo int.

```
int var1;
```

```
int var2;
```

```
int var3;
```

Outra forma também correta para a mesma declaração seria a que se segue

```
int var1, var2, var3;
```

Questão de concurso

5 Cespe TRT-ES 2009 O valor 3.1415F define uma variável do tipo long de 32 bits, inteira, com sinal em complemento de dois.

Questão de concurso

5 Cespe TRT-ES 2009 O valor 3.1415F define uma variável do tipo long de 32 bits, inteira, com sinal em complemento de dois.

Questão de concurso

6 FCC TCE-SP 2012 Em um programa Java, considere a existência de uma variável do tipo long chamada cod contendo o valor 1234. Para passar o valor contido nessa variável para uma variável do tipo byte chamada codNovo, deve-se fazer casting. Para isso, utiliza-se a instrução: byte codNovo =

- a) Byte.valueOf(cod);
- b) (long) cod;
- c) Byte.parseByte(cod);
- d) (byte) cod;
- e) (cast) cod;

Questão de concurso

6 FCC TCE-SP 2012 Em um programa Java, considere a existência de uma variável do tipo long chamada cod contendo o valor 1234. Para passar o valor contido nessa variável para uma variável do tipo byte chamada codNovo, deve-se fazer casting. Para isso, utiliza-se a instrução: byte codNovo =

- a) Byte.valueOf(cod);
- b) (long) cod;
- c) Byte.parseByte(cod);
- d) (byte) cod;
- e) (cast) cod;

Questão de concurso

7 ESAF CGU 2012 Em linguagem Java

- a) == significa atribuição. & significa “E” lógico. || significa “OU” lógico.
- b) == significa igualdade. && significa atribuição lógica. || significa “+” lógico.
- c) == significa igualdade. && significa “E” lógico. || significa “OU” lógico.
- d) <> significa igualdade. &+ significa “E” lógico. | significa “OU” lógico.
- e) += significa igualdade superior. && significa “E” lógico. |= significa “OU” lógico.

Questão de concurso

7 ESAF CGU 2012 Em linguagem Java

- a) == significa atribuição. & significa “E” lógico. || significa “OU” lógico.
- b) == significa igualdade. && significa atribuição lógica. || significa “+” lógico.
- c) == significa igualdade. && significa “E” lógico. || significa “OU” lógico.
- d) <> significa igualdade. &+ significa “E” lógico. | significa “OU” lógico.
- e) += significa igualdade superior. && significa “E” lógico. |= significa “OU” lógico.

Questão de concurso

8 FGV Badesc 2010

Observe o código em Java a seguir, em que se pode verificar a aplicação dos operadores de pré-decremento e pós-decremento.

```
public class Decrementa{  
    public static void main (string args{ }){  
        int m, n = 44;  
        m = --n;  
        m = n--;  
        system.out.println(m);  
        system.out.println(n);}}
```

Após a execução do código, as variáveis m e n exibirão, respectivamente, os valores:

a) 42 e 41. b) 42 e 42. c) 42 e 43. d) 43 e 42. e) 43 e 43.

Questão de concurso

8 FGV Badesc 2010

Observe o código em Java a seguir, em que se pode verificar a aplicação dos operadores de pré-decremento e pós-decremento.

```
public class Decrementa{  
    public static void main (string args{ }){  
        int m, n = 44;  
        m = --n;  
        m = n--;  
        system.out.println(m);  
        system.out.println(n);}}
```

Após a execução do código, as variáveis m e n exibirão, respectivamente, os valores:

a) 42 e 41. b) 42 e 42. c) 42 e 43. **d) 43 e 42.** e) 43 e 43.

Questão de concurso

9 ESAF Receita Federal 2012 Em programação Java, o comando while

- a) executa um bloco exclusivamente de comandos de atribuição.
- b) executa um bloco de comandos enquanto sua condição for verdadeira.
- c) executa um bloco de comandos até que sua condição seja verdadeira.
- d) equivale ao comando what-if.
- e) é idêntico ao comando do while.

Questão de concurso

9 ESAF Receita Federal 2012 Em programação Java, o comando while

- a) executa um bloco exclusivamente de comandos de atribuição.
- b) executa um bloco de comandos enquanto sua condição for verdadeira.
- c) executa um bloco de comandos até que sua condição seja verdadeira.
- d) equivale ao comando what-if.
- e) é idêntico ao comando do while.

Questão de concurso

10 FCC TST 2012 Considere o programa abaixo escrito na linguagem Java:

```
public class Programa  
{  
    public static void main(String[] args){  
        for(int i = 3, i<20, i+=2){  
            System.out.print((i%3) + " ");  
        }  
    }  
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

- a) 0 0 1 0 0 1 0 0 1
- b) 0 1 2 0 1 2 0 1 2
- c) 0 1 0 1 0 1 0 1 0
- d) 1 2 1 2 1 2 1 2 1
- e) 0 2 1 0 2 1 0 2 1

Questão de concurso

10 FCC TST 2012 Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
    public static void main(String[] args){
        for(int i = 3, i<20, i+=2){
            System.out.print((i%3) + " ");
        }
    }
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

- a) 0 0 1 0 0 1 0 0 1
- b) 0 1 2 0 1 2 0 1 2
- c) 0 1 0 1 0 1 0 1 0
- d) 1 2 1 2 1 2 1 2 1
- e) 0 2 1 0 2 1 0 2 1

i	i%3

Questão de concurso

10 FCC TST 2012 Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
    public static void main(String[] args){
        for(int i = 3, i<20, i+=2){
            System.out.print((i%3) + " ");
        }
    }
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

- a) 0 0 1 0 0 1 0 0 1
- b) 0 1 2 0 1 2 0 1 2
- c) 0 1 0 1 0 1 0 1 0
- d) 1 2 1 2 1 2 1 2 1
- e) 0 2 1 0 2 1 0 2 1

i	i%3
3	0

Questão de concurso

10 FCC TST 2012 Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
    public static void main(String[] args){
        for(int i = 3, i<20, i+=2){
            System.out.print((i%3) + " ");
        }
    }
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

- a) 0 0 1 0 0 1 0 0 1
- b) 0 1 2 0 1 2 0 1 2
- c) 0 1 0 1 0 1 0 1 0
- d) 1 2 1 2 1 2 1 2 1
- e) 0 2 1 0 2 1 0 2 1

i	i%3
3	0
5	2

Questão de concurso

10 FCC TST 2012 Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
    public static void main(String[] args){
        for(int i = 3, i<20, i+=2){
            System.out.print((i%3) + " ");
        }
    }
}
```

i	i%3
3	0
5	2
7	1

O resultado a ser informado ao usuário após a execução do programa acima é:

- a) 0 0 1 0 0 1 0 0 1
- b) 0 1 2 0 1 2 0 1 2
- c) 0 1 0 1 0 1 0 1 0
- d) 1 2 1 2 1 2 1 2 1
- e) 0 2 1 0 2 1 0 2 1

Questão de concurso

10 FCC TST 2012 Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
    public static void main(String[] args){
        for(int i = 3, i<20, i+=2){
            System.out.print((i%3) + " ");
        }
    }
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

- a) 0 0 1 0 0 1 0 0 1
- b) 0 1 2 0 1 2 0 1 2
- c) 0 1 0 1 0 1 0 1 0
- d) 1 2 1 2 1 2 1 2 1
- e) 0 2 1 0 2 1 0 2 1

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s
6	0	1	-1
	1	2	

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s
6	0	1	-1
	1	2	
	-1	3	

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s
6	0	1	-1
	1	2	
	-1	3	
	2	4	

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s
6	0	1	-1
	1	2	
	-1	3	
	2	4	
	-2	5	

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s
6	0	1	-1
	1	2	
	-1	3	
	2	4	
	-2	5	
	3	6	

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

n	r	c	s
6	0	1	-1
	1	2	
	-1	3	
	2	4	
	-2	5	
	3	6	
	-3	7	

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. b) -3 e 7. c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

11 FCC MPE-PE 2012 Analise o código Java a seguir:

```
public class Classe1 {  
    public static void main(String[] args) {  
        int n, r, c, s;  
        n = 6; r = 0; c = 1; s = -1;  
        while (c <= n) {  
            if (c % 2 == 0) {  
                r = r + c * s;  
            } else {  
                r = r + c;  
            }  
            c++;  
        }  
        System.out.println(r);  
    }  
}
```

Ao compilar e executar a Classe1, os valores finais nas variáveis r e c serão respectivamente:

a) 3 e 6. **b) -3 e 7.** c) -2 e 7. d) 4 e 6. e) -3 e 6.

Questão de concurso

12 ESAF CGU 2012 Na linguagem Java, o comando continue tem a função de

- a) fazer com que o comando de seleção seja inicializado.
- b) permitir realçar a posição de determinados comandos
- c) modificar a estrutura do loop, realçando procedimentos.
- d) fazer com que a continuidade da execução de um loop fique condicionada a um teste de condição de continuidade.
- e) fazer com que a condição do comando de loop seja novamente testada, mesmo antes de alcançar o fim do comando.

Questão de concurso

12 ESAF CGU 2012 Na linguagem Java, o comando continue tem a função de

- a) fazer com que o comando de seleção seja inicializado.
- b) permitir realçar a posição de determinados comandos
- c) modificar a estrutura do loop, realçando procedimentos.
- d) fazer com que a continuidade da execução de um loop fique condicionada a um teste de condição de continuidade.
- e) fazer com que a condição do comando de loop seja novamente testada, mesmo antes de alcançar o fim do comando.

Gabarito

1 b

2 e

3 c

4 c

5 e

6 d

7 c

8 d

9 b

10 e

11 b

12 e

4 Objetos e Classes

4.1 Introdução

- A programação orientada a objetos trabalha modelando as aplicações por meio de objetos do mundo real.
- Suas principais vantagens são:
 - Facilidade de manutenção
 - Reusabilidade
 - Extensibilidade

4.1 Introdução

- Objetos na vida real possuem atributos e realizam ações.



4.1 Introdução

- Objetos em Java também possuem atributos (ou campos) e ações (métodos) associados.
- As classes em Java são modelos para a construção de objetos.
- Se você tem uma classe **Empregado**, você pode construir quantos objetos **Empregado** quiser, por exemplo.
- A classe determina o objeto.

4.1 Introdução

- A abstração de se usar classes e objetos para modelar o mundo real não implica na obrigatoriedade de se modelar tudo.
- Em Java, as classes são os blocos fundamentais de todo o código construído.
- Até programas simples, que não necessitam das características da orientação a objetos, residem em classes.

4.2 Classes

- Uma classe necessita, inicialmente, de três coisas:
 - Um nome,
 - Atributos, e
 - Métodos.
- A definição de uma classe em Java tem a sintaxe:

```
1 class nome_da_classe {  
2 //corpo da classe  
3 }
```


4.2 Classes

- Exemplo 1:

```
1 class Empregado {  
2   int idade;  
3   double salario;  
4 }
```

- A definição de uma classe deve ser guardada em um arquivo com o mesmo nome da classe, com a extensão .java.

4.2 Classes

- Os atributos são variáveis, que podem ser de tipos primitivos ou referências a outros objetos.

```
1 class Empregado {  
2   int idade;  
3   double salario;  
4   Endereco endereco;  
5 }
```

4.2 Classes

- Métodos definem ações que um objeto (uma instância) pode fazer.
- Um método possui um cabeçalho e um corpo.
- O cabeçalho possui um tipo de retorno, o nome do método e uma lista de argumentos:

```
1 tipo_de_retorno Nome(argumentos) {  
2 //instrução(ões)  
3 }
```

4.2 Classes

- O tipo de retorno indica a saída do método: se vai ser um tipo primitivo, um objeto ou void (nada).
- O cabeçalho é conhecido como assinatura do método.
- A lista de argumentos são as entradas do método, informações que indicarão o fluxo de execução e que serão processadas.

4.2 Classes

- Exemplos:

```
1 double getSalario() {  
2     return salario;  
3 int soma(int a, int b) {  
4     return a + b;  
}
```

- Existe um método especial chamado **main** que é o ponto de partida da aplicação.

4.2 Classes

- A assinatura do método main:

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

- A lista de argumentos de main restringe-se a strings e pode ser passada na execução do programa java, por meio da linha de comando. Exemplo:

```
java Test 1 safeMode
```

4.2 Classes

- Toda classe deve ter ao menos um construtor (nem que seja um implícito).
- Ele é utilizado na construção de um objeto.
- Ele recebe o mesmo nome da classe.
- Sua sintaxe:

```
1 Nome(lista_de_argumentos) {  
2   //corpo do construtor  
3 }
```

4.2 Classes

- Um exemplo:

```
01 public class Empregado {  
02     public int idade;  
03     public double salario;  
04     public Empregado() {  
05     }  
06     public Empregado(int valorIdade, double  
07     valorSalario) {  
08         idade = valorIdade;  
09         salario = valorSalario;  
10     }  
11 }
```


4.3 Objetos

- Um objeto também é chamado de uma instância da classe.
- A forma mais comum de se criar um objeto é por meio de `new` + construtor da classe e atribuí-lo a uma variável de referência para o tipo de objeto que está sendo criado.
- Exemplo:

```
1 Empregado empregado = new Empregado();
```

4.3 Objetos

- Uma vez que você tenha um objeto você utiliza o `.` para chamar os métodos e acessar os atributos.

```
1 Empregado empregado = new Empregado();  
2 empregado.idade = 33;  
3 empregado.salario = 12000;
```

4.3 Objetos

- Uma variável de referência aponta para um objeto. Quando não se tem um objeto, ela aponta para null.
- Por exemplo, o atributo abaixo de uma classe qualquer receberá o valor null se na criação do objeto nenhum livro for associado ao atributo.

```
1 Livro livro;
```

4.3 Objetos

- Acessar um atributo de um objeto null dá erro!

```
1 System.out.println(livro.titulo);
```

- Você pode testar se uma variável de referência é null com o operador ==

```
1 if (livro == null) {  
2     livro = new Livro();  
3 }
```

4.4 Objetos na memória

- Quando declaramos variáveis é fácil calcular a quantidade de memória alocada. Para um int, por exemplo, temos 32 bits.
- Porém, para variáveis de referência a história é diferente.
- Quando rodamos um programa, o espaço de armazenamento de dados é dividido em dois: a pilha e a heap.

4.4 Objetos na memória

- Quando você executa o código abaixo, a variável de referência empregado recebe um espaço de memória na pilha onde será armazenado o endereço para o espaço de memória que foi alocado para o objeto que acabou de ser criado na heap.

```
1 Empregado empregado = new Empregado();
```

4.4 Objetos na memória

- Um objeto pode ser referenciado por mais de uma variável de referência:

```
1 Livro meulivro = new Livro();  
2 Livro seulivro = meulivro;
```

- O código abaixo cria dois objetos diferentes:

```
1 Livro meulivro = new Livro();  
2 Livro seulivro = new Livro();
```

4.4 Objetos na memória

- Quando um atributo de um objeto é uma referência a outro objeto, tem-se que na pilha temos uma variável de referência que aponta para o objeto na heap.
- Dentro do objeto na heap, tem-se uma variável de referência que aponta para o outro objeto que também está na heap.

```
1 public class Empregado{  
2     Endereco endereco = new Endereco;  
3 }
```


4.4 Objetos na memória

- Aprendemos a criar objetos com o new, mas em nenhum momento foi falado como destruir objetos sem uso para liberar memória.
- Java possui o garbage collector, que destrói objetos sem uso.
- Um objeto sem uso é aquele que não possui referências ou objetos que o referencia.

4.5 Pacotes

- Em Java, você pode agrupar classes com funcionalidades similares em pacotes.
- Pacotes que começam com java são reservados para a biblioteca padrão e os que começam com javax são extensões da biblioteca padrão.
- Além de organizar as classes, os pacotes também ajudam a resolver conflitos de nomes.

4.5 Pacotes

- Um pacote não é um objeto físico. Ele não precisa ser criado.
- Para agrupar uma classe num pacote, use `package` mais o nome do pacote em cima da definição da classe. Exemplo:

```
1 package com.empresa;  
2 public class Empregado{  
3     Endereco endereco = new Endereco;  
4 }
```

4.6 Encapsulamento e controle de acesso

- Encapsulamento é um princípio de orientação à objeto em que se protege partes do objeto que precisam ser protegidas e expõe partes que são seguras de serem expostas.
- Existem quatro modificadores de acesso:
 - public
 - private
 - protected
 - default

4.6 Encapsulamento e controle de acesso

- Controle de acesso para classes:
 - Uma classe pode ser pública ou default. Uma classe pública é vista em qualquer lugar.
 - Para declarar uma classe pública:

```
1 package com.empresa;  
2 public class Empregado{  
3     Endereco endereco = new Endereco;  
4 }
```

4.6 Encapsulamento e controle de acesso

- Controle de acesso para classes:
 - Ter um controle de acesso default significa não utilizar nenhum modificador de acesso na definição da classe.
 - Classes com controle de acesso default só podem ser vistas por outras classes que pertencerem ao mesmo pacote.

4.6 Encapsulamento e controle de acesso

- Controle de acesso para atributos, métodos e construtores.

Nível de acesso	Classes de outros pacotes	Classes do mesmo pacote	Subclasses	Mesma classe
public	sim	sim	sim	sim
protected	não	sim	sim	sim
default	não	sim	não	sim
private	não	não	não	sim

4.6 Encapsulamento e controle de acesso

- Exemplificando:

```
01 public class Empregado {
02     private int idade;
03     private double salario;
04     public Empregado() {
05     }
06     public Empregado(int valorIdade, double
07     valorSalario) {
08         idade = valorIdade;
09         salario = valorSalario;
10     }
11 }
```


4.7 this

- A palavra reservada `this` é utilizada dentro de um método ou construtor para referenciar o objeto atual.
- O construtor do slide anterior poderia ser reescrito da seguinte forma:

```
1 public Empregado(int idade, double salario)
2 {
3     this.idade = idade;
4     this.salario = salario;
5 }
```

4.8 Usando outras classes

- É comum querer criar objetos ou utilizar métodos de outras classes dentro da classe em desenvolvimento pelo programador.
- O uso de classes do mesmo pacote é permitido, mas para usar classes de outros pacotes é necessário importá-las.
- Para usar a classe `java.io.File`, por exemplo, é preciso:

```
1 import java.io.File;
```

4.8 Usando outras classes

- Pode-se ter vários import.
- Pode-se também importar todas as classes de um pacote por meio do *.
- É possível também importar constantes.

```
1 package xpto;  
2 import java.io.*;  
3 import java.util.List;  
4 import static java.util.Calendar.SATURDAY;  
5 public class ...
```

4.8 Usando outras classes

- A única forma de se utilizar uma classe de outro pacote sem o import é por meio do “fully qualified name”:

```
1 java.io.File file = new java.io.File(nome);
```

- Quando se importar classes com mesmo nome de diferentes pacotes, o “fully qualified name” também é necessário.

4.9 Métodos estáticos

- Relembrando, para usar um método de uma classe específica, é preciso criar um objeto primeiro:

```
1 Livro meulivro = new Livro();  
2 meulivro.ler();
```

- Porém, alguns métodos (como o `System.out.println()`) não precisam da criação de um objeto. Eles são chamados de métodos estáticos.

4.9 Métodos estáticos

- Para criar um método estático basta adicionar a palavra reservada `static` ao cabeçalho.

```
1 public class MathUtil {  
2     public static int soma(int a, int b) {  
3         return a + b;  
4     }  
5 }
```

- Depois, é só usar o método em outra classe.

```
1 int a;  
2 a = MathUtil.soma(2, 2);
```

4.10 Atributos estáticos

- Uma variável declarada com a palavra `static` torna-se um atributo de classe e não de objeto. Ou seja, os objetos criados compartilharão a variável entre eles.

```
1 public class Livro{  
2     private static int copia;  
3     public Livro ( ){  
4         copia++;  
5     }  
6 }
```

4.11 Escopo de variáveis

- Vimos que podemos declarar variáveis em diversos locais:
 - Em uma classe como atributo.
 - Como parâmetro de um método ou construtor.
 - Dentro do método ou construtor.
 - Dentro de um bloco de uma instrução, como um for ou while.

4.11 Escopo de variáveis

- O escopo de uma variável indica onde ela está acessível.
- Regra 1: a variável só existe dentro do bloco em que foi criada.
- No exemplo abaixo, x só existe dentro do for. Quando o for termina, a JVM destrói x.

```
1 for (int x = 0; x < 5; x++) {  
2   System.out.println(x);  
3 }
```

4.11 Escopo de variáveis

- Regra 2: blocos internos podem acessar variáveis declaradas no bloco externo.

Exemplo:

```
1 for (int x = 0; x < 5; x++) {  
2   for (int y = 0; y < 3; y++) {  
3     System.out.println(x);  
4     System.out.println(y);  
5   }  
6 }
```

4.11 Escopo de variáveis

- Regra 3: Variáveis declaradas como parâmetros em métodos podem ser acessadas pelo corpo do método.
- Regra 4: Variáveis no nível de classe (atributos) são acessadas em toda a classe.
- Regra 5: Se uma variável dentro de um método tem o mesmo nome de um atributo da classe, a variável do método irá se “sobrepor” ao atributo da classe.

4.12 Overloading

- Java possibilita o uso do mesmo nome em diferentes métodos com funcionalidades semelhantes, desde que cada método tenha uma lista de argumentos diferente. O tipo de retorno não é levado em consideração.
- Exemplo:

```
1 int soma(int a, int b) {...}  
2 float soma(float a, float b) {...}  
3 int soma(int a, int b, int c) {...}
```

4.13 Passagem de parâmetros

- Em programação, falamos em passagem de parâmetros, quando uma variável é utilizada como argumento para um método.
- Variáveis primitivas são passadas por valor, ou seja, uma cópia do valor armazenado na variável é passado para dentro do método, qualquer mudança do valor internamente no método não refletirá na variável.
- Variáveis de referência são passadas por referência, ou seja, o endereço do objeto é passado para dentro do método e o objeto será modificado pelas ações do método.
- Veja o exemplo a seguir.

```
01 package teste;
02 class Ponto {
03     public int x; public int y; }
04 public class ReferencePassingTest {
05     public static void increment(int x) {
06         x++; }
07     public static void reset(Ponto ponto) {
08         ponto.x = 0; ponto.y = 0; }
09     public static void main(String[] args) {
10         int a = 9;
11         increment(a);
12         System.out.println(a); // 9
13         Ponto p = new Ponto();
14         p.x = 400;
15         p.y = 600;
16         reset(p);
17         System.out.println(p.x); // 0 } }
```

Questão de concurso

1 Cespe BASA 2012 Em Java, para toda classe, método e variável de instância que se declara há um controle de acesso, independentemente de o controle ser explicitamente indicado.

Questão de concurso

1 Cespe BASA 2012 Em Java, para toda classe, método e variável de instância que se declara há um controle de acesso, independentemente de o controle ser explicitamente indicado.

Questão de concurso

2 Cespe INPI 2012 Uma classe pode acessar atributos de uma superclasse mesmo quando eles são declarados como private.

Questão de concurso

2 Cespe INPI 2012 Uma classe pode acessar atributos de uma superclasse mesmo quando eles são declarados como private.

Questão de concurso

3 Cespe BASA 2012 Os métodos de uma classe Java criada com controle de acesso protect podem ser acessados apenas por classes do mesmo pacote.

Questão de concurso

3 Cespe BASA 2012 Os métodos de uma classe Java criada com controle de acesso protect podem ser acessados apenas por classes do mesmo pacote.

Questão de concurso

4 FCC TRE-CE 2012

```
public class calculo {  
    public static double soma(double n1, double n2){...}  
    public static double soma(double n1, double n2, double n3){...}  
}
```

O método soma representa um exemplo de

- a) sobrecarga de métodos.
- b) herança de parâmetros.
- c) recursividade.
- d) encapsulamento.
- e) sobrecarga de métodos.

Questão de concurso

4 FCC TRE-CE 2012

```
public class calculo {  
    public static double soma(double n1, double n2){...}  
    public static double soma(double n1, double n2, double n3){...}  
}
```

O método soma representa um exemplo de

- a) sobrecarga de métodos.
- b) herança de parâmetros.
- c) recursividade.
- d) encapsulamento.
- e) sobrecarga de métodos.

Questão de concurso

5 FCC MPE-PE 2012 Sobre a programação orientada a objetos com Java, é correto afirmar:

- a) Uma classe pode ter mais de um método com o mesmo nome, desde que receba parâmetros diferentes.
- b) É obrigatório escrever em toda classe um construtor vazio que não recebe parâmetros. Além desse construtor, pode-se criar outros construtores vazios.
- c) Em uma hierarquia de herança, a superclasse herda todos os métodos públicos, privados e protegidos das subclasses.
- d) Uma interface possui métodos com conteúdo implementado que não precisam ser reescritos nas classes que a implementam.
- e) No interior do método main de uma classe, mesmo os métodos públicos e estáticos de outra classe só podem ser acessados por meio de um objeto explicitamente instanciado dessa classe.

Questão de concurso

5 FCC MPE-PE 2012 Sobre a programação orientada a objetos com Java, é correto afirmar:

- a) Uma classe pode ter mais de um método com o mesmo nome, desde que receba parâmetros diferentes.
- b) É obrigatório escrever em toda classe um construtor vazio que não recebe parâmetros. Além desse construtor, pode-se criar outros construtores vazios.
- c) Em uma hierarquia de herança, a superclasse herda todos os métodos públicos, privados e protegidos das subclasses.
- d) Uma interface possui métodos com conteúdo implementado que não precisam ser reescritos nas classes que a implementam.
- e) No interior do método main de uma classe, mesmo os métodos públicos e estáticos de outra classe só podem ser acessados por meio de um objeto explicitamente instanciado dessa classe.

Questão de concurso

6 Cesgranrio CMB 2009

```
public class Main {  
    public static void main(String[] args){  
        int x = 1; int y = 3; int z = 6;  
        x += calcula(y++ + ++z);  
        system.out.println(x);}  
    public static int calcula(int x){  
        return x*2;}  
    public static int calcula(int y, int z){  
        return z-y;}
```

Qual será a saída do programa Java mostrado acima?

a) 3. b) 5. c) 19. d) 21. e) 23.

Questão de concurso

6 Cesgranrio CMB 2009

```
public class Main {  
    public static void main(String[] args){  
        int x = 1; int y = 3; int z = 6;  
        x += calcula(y++ + ++z);  
        system.out.println(x);}  
    public static int calcula(int x){  
        return x*2;}  
    public static int calcula(int y, int z){  
        return z-y;}
```

Qual será a saída do programa Java mostrado acima?

a) 3. b) 5. c) 19. **d) 21.** e) 23.

Gabarito

1 c

2 e

3 e

4 e

5 a

6 d

Fim