EN05170 - Programação de Computadores II

Introdução a Programação Orientada a Objetos – Sintaxe Java

Prof. Dr. Lidio Mauro Lima de Campos

limadecampos@gmail.com

Universidade Federal do Pará – UFPA
ICEN
PPGCC

Agenda

- Introdução
- Iniciando
- Uso de Maiúsculas e minúsculas
- Comentários
- Tipos Primitivos
- Identificadores, Keywords e Tipos
- Blocos de Código e uso de ;
- Constantes, Variáveis, declarações, atribuições
- Escopo e Tempo de Vida de Variáveis
- Entradas e Saídas Básicas
- Operadores e expressões

Introdução

Programa mínimo

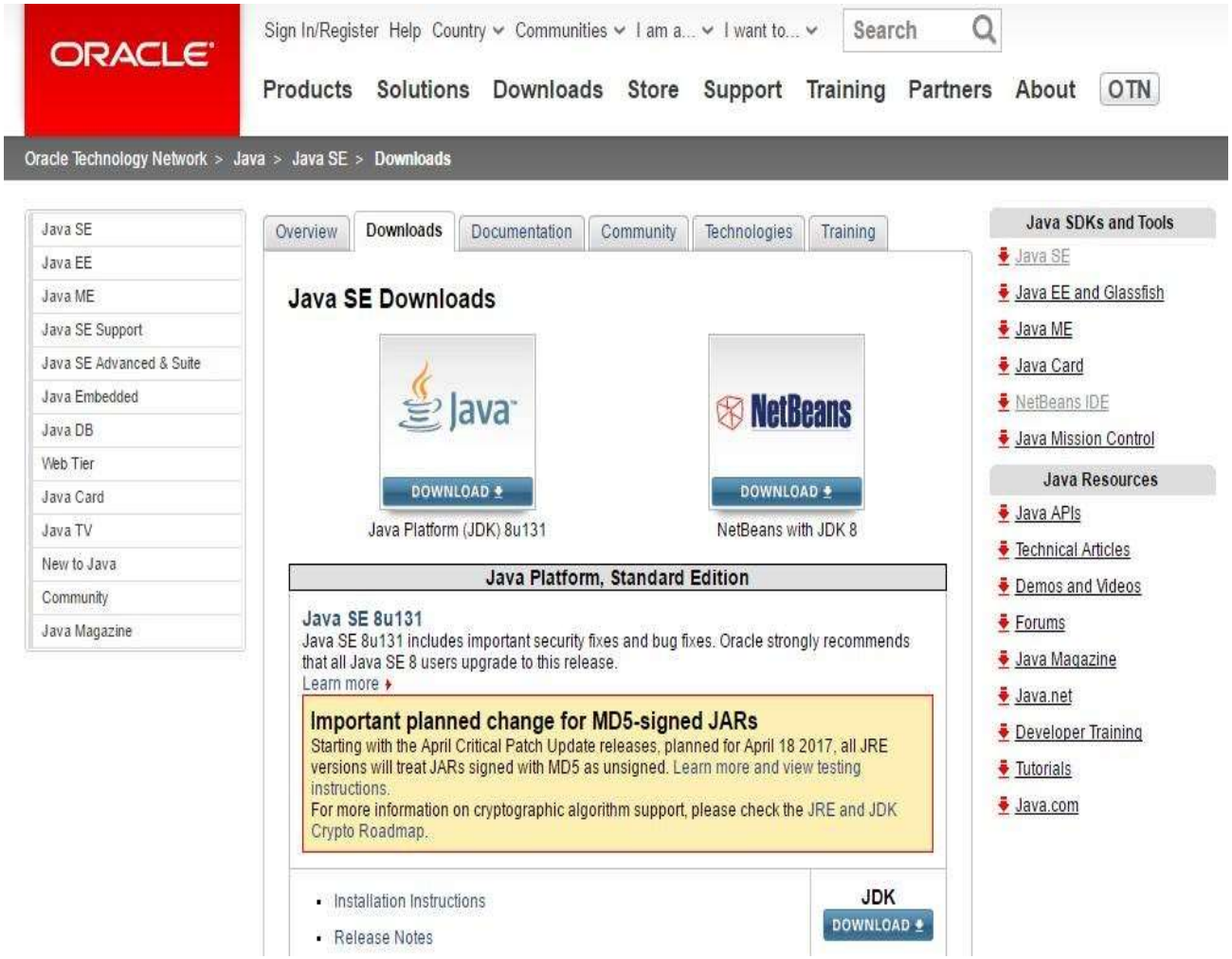
Um programa em Java pode ser composto de um ou mais arquivos-fonte, denominados unidades de compilação, os quais podem conter:

- Uma declaração de pacote (package);
- Uma ou mais diretivas de importação (import);
- Uma ou mais declarações de classes (class), de interfaces (interface) ..

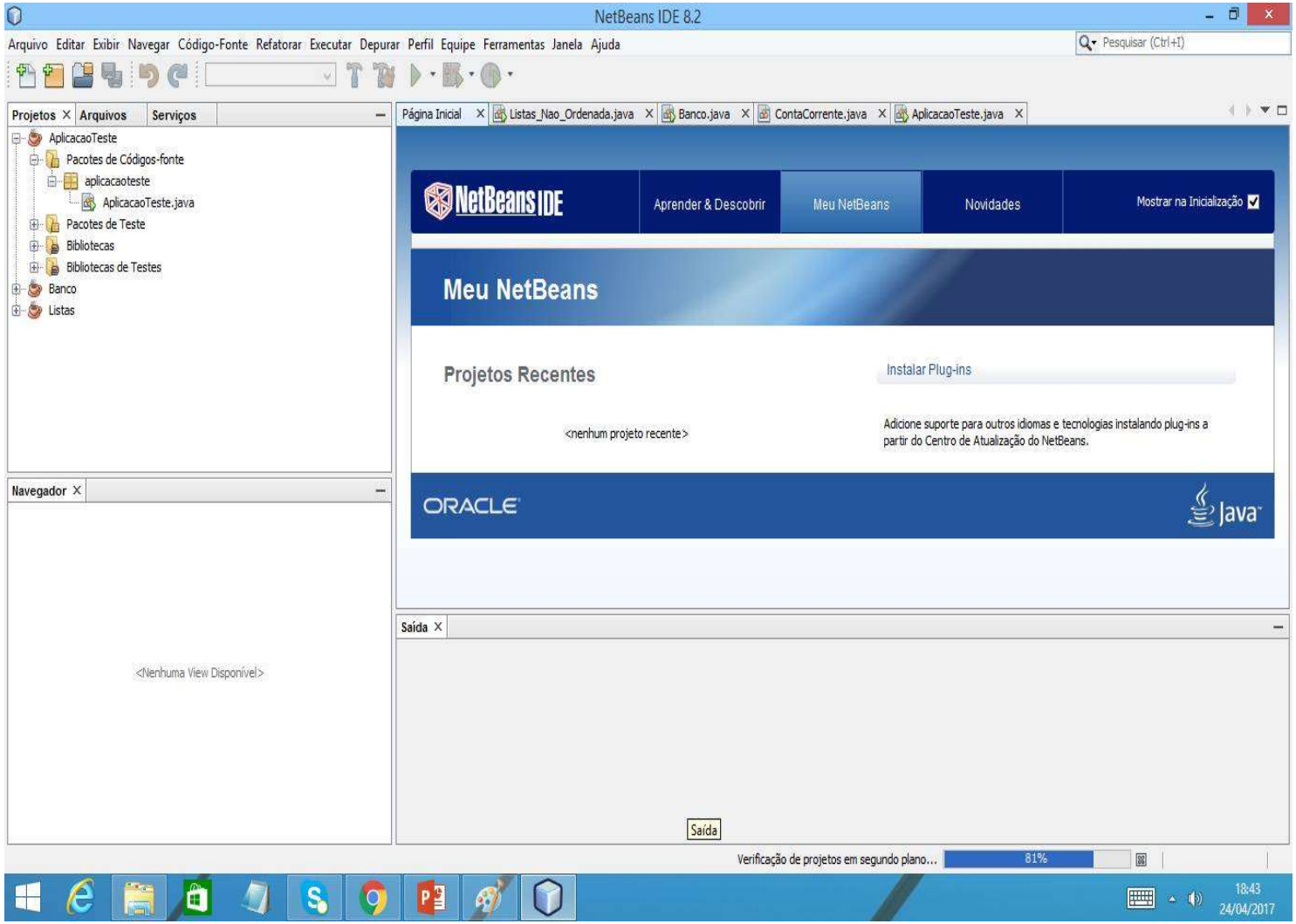
- Todo programa deve ter, no mínimo, o método main(String[]) que define seu inicio e é declarado public, static e void dentro de alguma de suas classes.

NetBeans IDE 14 com JDK

- Obter em: https://netbeans.apache.org/download/index.html



NetBeans IDE 14 com JDK

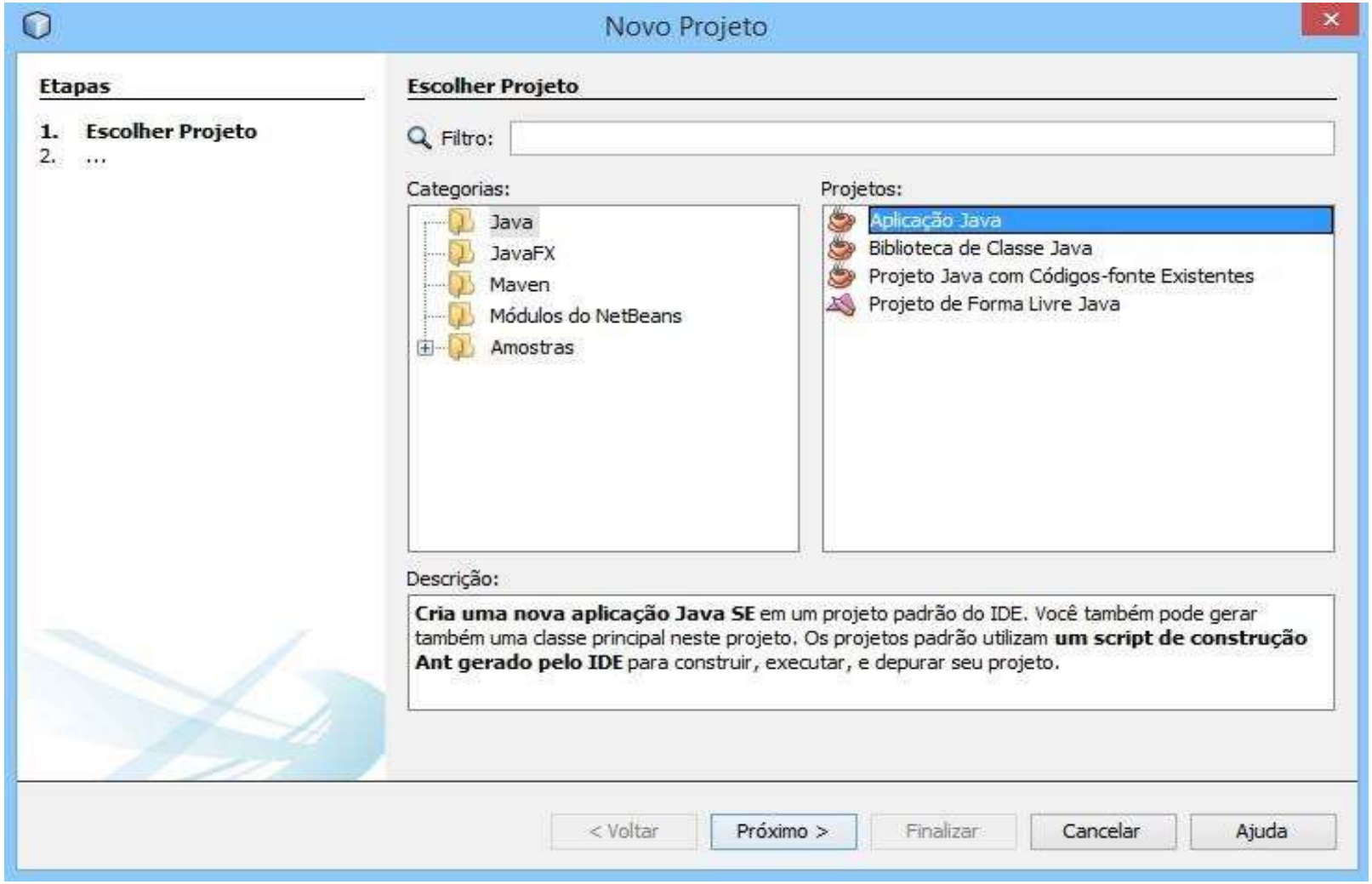


jdk-8u131-nb-8_2-windows-x64

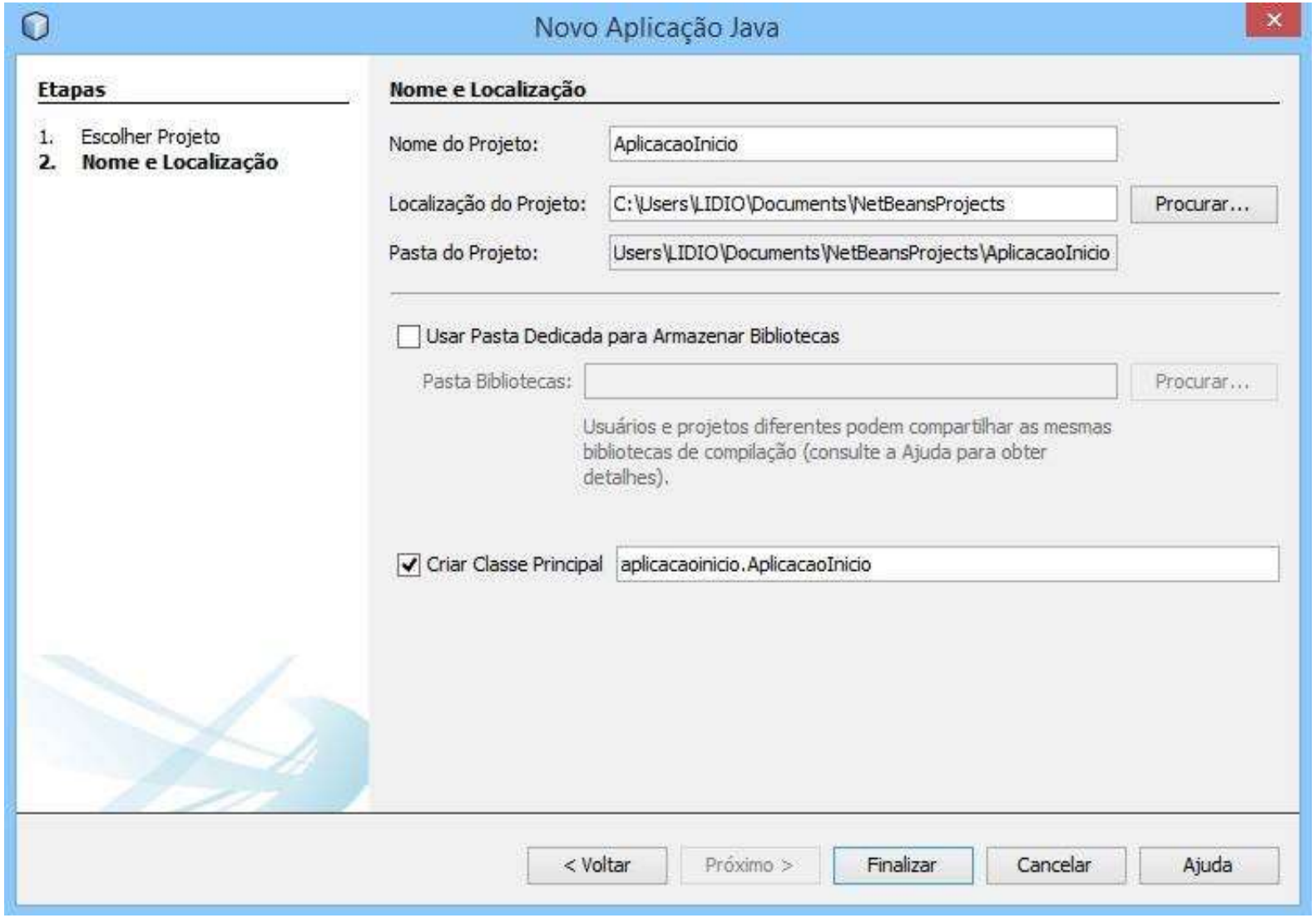
jre-8u121-windows-x64

Iniciando

- No NetBeans, File , New Project



Iniciando


Novo Aplicação Java
Etapas
1. Escolher Projeto
2. Nome e Localização
Nome e Localização
Nome do Projeto:
AplicacaoInicio
Localização do Projeto:
C:\Users\LIDIO\Documents\NetBeansProjects
Procurar...
Pasta do Projeto:
Users\LIDIO\Documents\NetBeansProjects\AplicacaoInicio
Usar Pasta Dedicada para Armazenar Bibliotecas
Pasta Bibliotecas:
Procurar...
Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).
Criar Classe Principal
aplicacaoinicio.AplicacaoInicio
< Voltar
Próximo >
Finalizar
Cancelar
Ajuda

Iniciando

```
public class AplicacaoInicio
{  public static void main(String args[])
   { System.out.println("oi");
     String nome="lidio";
     System.out.println("Meu nome:"+nome);
     int idade=30;
     System.out.println("Idade:"+idade);
     boolean log=true;
     System.out.println("Logico:"+log);
     short a=-32768;
     System.out.println("short:"+a);
     long b = 92233720368547758O7L;
     System.out.println("long:"+b);
     byte c=-128;
     System.out.println("byte:"+c);
     double x=5.67e-12;
     System.out.println("double:"+x);
     float y=8.78f;
     System.out.println("float:"+y);}
}
```

Uso de Maiúsculas e minúsculas

- Java é uma linguagem sensível ao contexto, em todas as suas construções as letras maiúsculas e minúsculas são consideradas diferentes, ou seja : nome é diferente de NOME, public é diferente de Public.

Comentários

Delimitador	Descrição
//	Comentário de linha
/* */	Comentário de bloco (Múltiplas linhas)

- float k=19.5f; // variável float iniciada com valor apropriado

- /* Declaração de duas variáveis
- Uma é inicializada com um valor arbitrário e a outra
- Recebe o dobro da primeira. */

- double a=10.3;
- double b=2*a;

Tipos de dados primitivos

Categoria	Tipo	Bytes/bits	Faixa de valores
Inteiro	byte (inteiro de oito bits)	1/8	-128 a 127
	short (Inteiro Curto)	2/16	-32768 a 32767
	int (inteiro)	4/32	-2.147.483.648 a 2.147.483.647
	long (inteiro longo)	8/84	-9.223.372.036.854.775.808L a 9.223.372.036.854.775.807L
Real	float (Precisão simples)	4/32	Valores Positivos +1.40129846432481707e-45 a +3.40282346638528860e+38 Valores Negativos -3.40282346638528860e+38 a -1.40129846432481707e-45
	double (Precisão Dupla)	8/64	Valores Positivos +4.94065645841246544e-324 a +1.79769313486231570e+308 Valores negativos -1.79769313486231570e+308 a -4.94065645841246544e-324
Caracter	char	2/16	\u0000 a \UFFFF
Lógico	boolean	1/8	false e true

Tipos de dados primitivos

Tabela 2.1 – Tipos de dados primitivos

Categoria	Tipo	Bytes/bits	Faixa de valores
Inteiro	byte (inteiro de oito bits)	1/8	-128 a 127
	short (Inteiro Curto)	2/16	-32768 a 32767
	int (inteiro)	4/32	-2.147.483.648 a 2.147.483.647
	long (inteiro longo)	8/84	-9.223.372.036.854.775.808L a 9.223.372.036.854.775.807L
Real	float (Precisão simples)	4/32	Valores Positivos +1.40129846432481707e-45 a +3.40282346638528860e+38 Valores Negativos -3.40282346638528860e+38 a -1.40129846432481707e-45
	double (Precisão Dupla)	8/64	Valores Positivos +4.94065645841246544e-324 a +1.79769313486231570e+308 Valores negativos -1.79769313486231570e+308 a -4.94065645841246544e-324
Caracter	char	2/16	\u0000 a \UFFFF
Lógico	boolean	1/8	false e true

Tipos de dados primitivos

- Tipos de dados inteiros

-13	27	7012	304586	1234567890

Não existe em Java o modificador unsigned (disponível em outras linguagens), assim os tipos inteiros são sempre capazes de representar tantos valores positivos como negativos.

Tipos de dados primitivos

- Tipos de dados inteiros

```
class testalong
{
 public static void main(String args[])
 {
  long ci;  long im;

  im = 5280 * 12; //  63360

  ci = im * im * im; //254358061056000

  System.out.println(“ci = " + ci );
 }
}
```

Tipos de dados primitivos

- Tipos de dados reais

0.123456789	1.44E6	3.4254e-2	-25.342E8	0.32E-17

O caractere ponto . Deve ser usado como separador de casas decimais, enquanto expoentes podem ser escritos usando o caractere e ou E.

Tipos de dados primitivos

- Tipos de dados reais

```
class Hypot {
 public static void main(String args[])
 {double x, y, z;
  x = 3;
  y = 4;

  z = Math.sqrt(x*x + y*y);
  System.out.printf("Hypotenuse is = %.2f\n ",z);
  System.out.println("Hypotenuse is " +z);
 }
}
```

Consultar API : Pacote Java lang

https://docs.oracle.com/javase/7/docs/api/

Tipos de dados primitivos

- Tipos de dados lógicos
- O Java dispõe do tipo lógico boolean capaz de assumir valores falso ou verdadeiro, os quais equivalem aos valores literais false e true

Não há equivalência entre valores lógicos (boolean) e inteiros (byte, short, int ou long), assim o valor zero não é tomado como false e valores diferentes de zero não são avaliados como true.

Tipos de dados primitivos

- Representa os valores verdadeiro/falso. Java define os valores verdadeiro e falso usando as palavras reservadas **true** e **false**.

```
class BoolDemo
{
 public static void main(String args[])
 { boolean b; b = false;
  System.out.println("b is " + b);  b = true;
  System.out.println("b is " + b);

  if(b) System.out.println("Isso é Executado.");
  b = false;
  if(b) System.out.println("Isso não é executado.");
  System.out.println("10 > 9 is " + (10 > 9));
 }
}
```

Tipos de dados primitivos

- Tipos de dados caractere

O tipo char representa caracteres individuais empregando um formato interno no padrão Unicode, em que cada caractere ocupa 8 ou 16 bits sem sinal, o que permite diferenciar até 32768 caracteres. O valor literal de caracteres é sempre delimitado por aspas simples (‘):

‘A’	‘c’	‘7’	‘\n’	‘\U0044A’

Tipos de dados primitivos

- Tipos de dados caractere

- https:// www.tamasoft.co.jp/en/general-info/unicode.html

```
public class Unicode {
  public static void main(String args[])
  {
   char ch;  ch = 'X';
   System.out.println("ch contains "+ch);

   ch++; // increment ch
   System.out.println("ch is now " +ch);

   ch = 90; // give ch the value Z
   System.out.println("ch is now " +ch);
 }
}
```

Tipos de dados primitivos

- Tipos de dados cadeia de caracteres – A combinação de um ou mais caracteres em sequencia é uma cadeia de caracteres ou uma String, representada no Java por objetos da classe String, ou seja, não é um tipo primitivo. Seus valores literais são delimitados por aspas duplas (“).

“J”	“Java”	“Java-Guia do Programador”	“oi\n turma”

- String comum=“Um texto dividido\n”
  +”em várias linhas, mas\n”
  +”Com caracteres especiais.”;
- String multilinha=“””
  Um texto dividido
  em várias linhas, mas
  SEM caracteres especiais.
  “””;

Strings não são Tipos Primitivos

- Em Java, strings não são tipos primitivos, são objetos!
- String a = “UFPA”;
- Concatenação: “ Hello” + “World” = “ HelloWorld”
- Conversão Implícita: “ Hello” + 2001 =“Hello2001”;
- Comparação de Strings: **a.equals(b) ;**
- Tamanho de uma String: a.length() ;

Variáveis

- Uma variável é um nome definido pelo programador ao qual se associa um valor que pertence a um tipo de dados particular, armazenado em um conjunto de posições de memória do computador.
- Possui um nome, um tipo e um valor.
- O nome é formado por:
  - Uma sequencia de um ou mais caracteres alfabéticos e numéricos, iniciados por uma letra ou pelos caracteres _(undescore) ou $(cifrão).
  - Nomes não podem conter símbolos gráficos, operadores ou espaços em branco, apenas os primeiros 32 caracteres são usados para distingui-los.
  - Exemplos válidos:

a	Total	X2	$mine	_especial

  - Exemplos não válidos:

1x	Total Geral	Numero-Minimo	void	super

Declaração e Inicialização de Variáveis

- Em Java, todas as variáveis são declaradas antes do seu uso.

  <tipo> <nome1> [, nome2, [nome3 [....nomeN]]];

  - *int count;*
  - *char ch;*
  - *float f;*

  tipo var=valor;

  - *int count=10;*
  - *char ch='X';*
  - *float f=1.2F;*
  - *int a,b=8,c=19,d;*

double raio=4, altura=5;

double volume=3.1416*raio*raio*altura;

Palavras Reservadas no Java

- A Linguagem Java possui 50 palavras reservadas.

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	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

Constantes, Variáveis, declarações, atribuições

```
public class Constantes
{
 static final double LARGURA = 10.0;

 public static void main(String[] args)
 {
   //final double LARGURA = 10.0;  double compr = 25.0;
   double res = calculaArea(LARGURA, compr);
  System.out.println("A área é: " + res);
 }
  private static double calculaArea(double largura, double  comprimento)
  {
     return largura * comprimento;
  }

}
```

Variáveis, declarações, atribuições

```
public class Variaveis
{ public static void main(String args [])
{ int x, y;  //declara duas variáveis inteiras
 float z = 3.414f; //declara e atribui um float
 double w = 3.1415; //declara e atribui um double
 boolean truth = true; //declara e atribui um boolean
 char c; //declara uma variável char
 String str; //declara uma String
 String str1 = “ufpa"; // declara e associa uma String
 c = 'S'; // declara e associa um char
  str = "Hello World!";
 x=6; y = 1000;// Associa uma valor a um int
System.out.println("x="+x);    System.out.println("y="+y);
 System.out.println("z="+z);  System.out.println("w="+w);  System.out.println(" truth="+truth);
System.out.println("c="+c); System.out.println(" str="+str); System.out.println("str1="+str1);
} }
```

Inferências de variáveis locais

- Java 10 introduziu um mecanismo que permite a determinação implícita do tipo de variável local pelo compilador, baseado no tipo de valor atribuído à variável, o que se denomina inferência (de tipo) de variável.

- **Declaração explicita :** double temperatura=21.5;
- **Mas agora é possível escrever**: var temperatura=21.5;

- A construção var se destina a indicar quais variáveis terão seus tipos inferidos (ou determinados) pelo compilador, baseado nos valores atribuídos a tais variáveis, sua sintaxe é:

- var <id1> [=expressao1][, <id2>[=expressoa2]..];

Blocos de Código e uso de ;

- Um *statement* é uma linha de código terminada por uma ";"

```
total = a + b + c;
```

- Um bloco de código é delimitado por chaves

```
{
 total = a + b + c;
}
```

Constantes em Java

```
public class UsaConstantes
{
   public static void main(String args[])
    {  final double CONST_ADIC=2.54;
       double largpapel=8.5;
       double altpapel=11;
       System.out.println("Tamanho do papel em centimetros:"+largpapel*CONST_ADIC+" por "+ altpapel*CONST_ADIC);
   }
}
```

Escopo de Variáveis

- O conjunto de locais no qual uma declaração tem validade é denominado escopo. Um bloco de comandos é um conjunto de comandos da linguagem delimitados por { }.

```
public class Escopo
{//inicio do bloco 0
  public static void main(String args[])
  {//inicio do bloco 1
     int i=5; //variavel do bloco 1
    {//inicio do bloco 2
     int j=0; //variavel do bloco 2
     System.out.println(i); //Uso do 1 dentro do seu escopo
     j=5*i; //Uso do j dentro do seu escopo
     System.out.println(j);
     {//inicio do bloco 3
      System.out.println(i);//Uso de i dentro do seu escopo
      System.out.println(j);//Uso de j dentro do seu escopo
     }
    }
     System.out.println(i); //Uso de i dentro do seu escopo
     System.out.println(j); //Uso de j FORA do seu escopo
  }
```

Escopo de Variáveis

```
class ScopeDemo {
public static void main(String  args[])
{ int x; // conhecida pelo código dentro de main
  x = 10;
 if(x == 10)
 {// começa novo escopo
   int y = 20; // conhecida apenas nesse bloco

   // x and y são conhecidas aqui  System.out.println("x and y: " + x + " " + y);

   x = y * 2;
}
// y = 100; // Erro! y não é conhecida aqui

// x ainda é conhecido aqui
System.out.println("x is " +x);
}
}
```

Escopo de Variáveis

```
class NestVar
{
 public static void main(String args[])
 {
  int count; // variável declarada com escopo externo  for(count = 0; count < 10;
  count = count+1)
   {
   System.out.println("Este Contador: " + count);

    int count; // ilnválido!!! , declarada novamente com escopo interno

    for(count = 0; count < 2; count++)
      System.out.println("Esse programa possui um erro!");
    }
  }
}
```

Escopo de Variáveis

- Variáveis local

–Declaradas dentro de um bloco de código { };
–Criadas quando o bloco de código inicia, destruídas quando ele acaba;
–DEVEM ser inicializadas ou o compilador emitirá um erro!

```
public static void main( String args[] )
{
int a = 10;
}
```

Entrada e Saídas Básicas

- Entrada
- A classe Scanner possui vários métodos que possibilitam a entrada de dados de diferentes tipos, entre eles destacam-se:

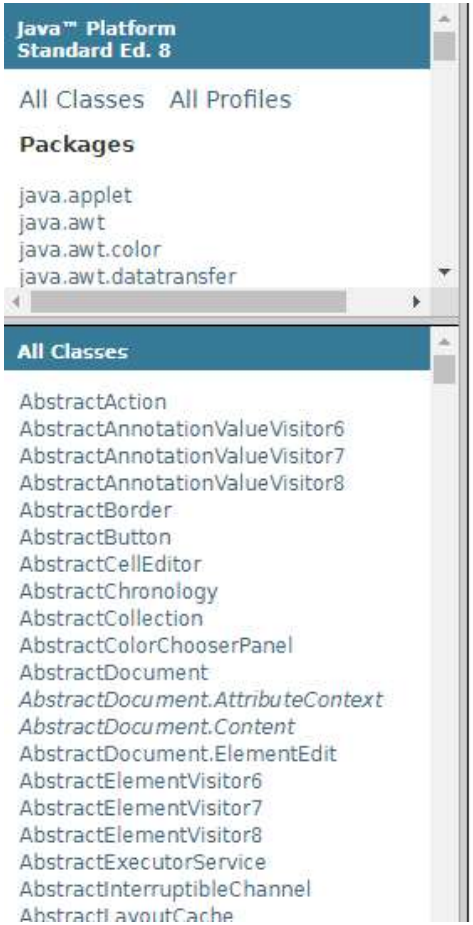
Método	Uso
String next()	retorna uma cadeia de caracteres simples, ou seja, que não usa o caractere espaço em branco;
double nextDouble()	retorna um número em notação de ponto flutuante normalizada em precisão dupla de 64 bits (usado para receber valores reais ou monetários);
int nextInt()	retorna um número inteiro de 32 bits;
String nextLine()	retorna uma cadeia de caracteres, por exemplo: “Programação II”;
long nextLong()	retorna um número inteiro de 64 bits.

Entrada e Saídas Básicas

- Entrada
- Para utilizar a classe Scanner em uma aplicação Java deve-se proceder da seguinte maneira:
- **1. Importar o pacote java.util:**
  - import java.util.Scanner;
- **2.Instanciar e criar um objeto Scanner usando o dispositivo padrão de entrada (System.in)**
  - Scanner ler = new Scanner(System.in);
- **3. Utilizar os métodos da classe Scanner adequados aos tipos das variáveis envolvidas. Os exemplos de entradas de dados serão demonstrados usando as seguintes variáveis:**
  - int n; double preco;
  - String palavra;
  - String frase;

Entrada e Saídas Básicas

- API JAVA – Pacote java.util https://docs.oracle.com/javase/8/docs/api/



java.security.interfaces	Provides interfaces for generating RSA (Rivest, Shamir and Adleman AsymmetricCipher algorithm) keys as defined in the RSA Laboratory Technical Note PKCS#1, and DSA (Digital Signature Algorithm) keys as defined in NIST's FIPS-186.
java.security.spec	Provides classes and interfaces for key specifications and algorithm parameter specifications.
java.sql	Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language.
java.text	Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.
java.text.spi	Service provider classes for the classes in the java.text package.
java.time	The main API for dates, times, instants, and durations.
java.time.chrono	Generic API for calendar systems other than the default ISO.
java.time.format	Provides classes to print and parse dates and times.
java.time.temporal	Access to date and time using fields and units, and date time adjusters.
java.time.zone	Support for time-zones and their rules.
java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).
java.util.concurrent	Utility classes commonly useful in concurrent programming.
java.util.concurrent.atomic	A small toolkit of classes that support lock-free thread-safe programming on single variables.

Entrada e Saídas Básicas

- API JAVA – Pacote java.util.Scanner https://docs.oracle.com/javase/8/docs/api/

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

As another example, this code allows long types to be assigned from entries in a file myNumbers:

```
Scanner sc = new Scanner(new File("myNumbers"));
while (sc.hasNextLong()) {
    long aLong = sc.nextLong();
}
```

The scanner can also use delimiters other than whitespace. This example reads several items in from a string:

```
String input = "1 fish 2 fish red fish blue fish";
Scanner s = new Scanner(input).useDelimiter("\\s*fish\\s*");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
s.close();
```

Entrada e Saídas Básicas

- Entrada : Scanner ler=new Scanner(System.in);
- Lendo um valor inteiro:
  - System.out.printf("Informe um número para a tabuada:\n");
  - Int n = ler.nextInt();
- Lendo um valor real:
  - System.out.printf("Informe o preço da mercadoria:\n");
  - Double preco = ler.nextDouble();
- Lendo uma String:
  - System.out.printf("Informe uma palavra:\n");
  - String palavra = ler.next();
- Lendo uma String:
  - System.out.printf("Informe uma frase:\n");
  - String frase = ler.nextLine();

Entrada e Saídas Básicas

- Saída
- A saída de dados para o console exibe (ou imprime) informações em modo texto, como segue:

- System.out.printn(“Oi!”);

    - System – Classe que disponibiliza um objeto out para qualquer aplicação;

    - out – Objeto que representa o dispositivo de saída dedados do console (sua janela);

    - println – método que lê que imprime valores, mensagens e objetos.

Entrada e Saídas Básicas

- Saída

- System.out.println(“1234”); //imprime um valor inteiro
- System.out.println(2.36); //imprime valor real
- System.out.println(“Palavra”);
- System.out.println(“Uma frase com palavras”);
- int numero=1234;
- System.out.println(numero);//imprime valor da variável
- System.out.println(numero*10+33);//imprime o resultado de uma expressão

- System.out.printf("%d / %d = %3d (divisão inteira)\n", a, b, (a / b));
- System.out.printf("%d / %d = %6.2f (divisão exata)\n", a, b, ((double)a / b));

Entrada e Saídas Básicas

- Saída
- int a, b;
- ....
- System.out.printf("%d / %d = %3d (divisão inteira)\n", a, b, (a / b));
- System.out.printf("%d / %d = %6.2f (divisão exata)\n", a, b, ((double)a / b));

- int x=10,y=3;
- System.out.printf("%d / %d = %6.2f (divisão exata)\n", x, y, ((float)x / y));

Exercícios

- 1)Implementar um programa em Java que leia dois valores inteiros e calcule e imprima as quatro operações básicas (soma, subtração, multiplicação e divisão).

Esvaziando o buffer do teclado

```
Scanner ler = new Scanner(System.in);
 // 1. instanciando e criando um objeto Scanner
 int idade;
 String nome;

 System.out.printf("Informe a sua idade:\n");
 idade = ler.nextInt(); // 2. entrada de dados (lendo um valor inteiro)

 ler.nextLine(); // esvazia o buffer do teclado

 System.out.printf("\nInforme o seu nome:\n");
 nome = ler.nextLine(); // 3. entrada de dados (lendo uma String)
```

Convenções do Java

- Java é case sensitive, Só por essa razão temos um bom motivo para padronizarmos como deveremos escrever nossos códigos;

- Além de tudo é uma boa prática toda a equipe escrever da mesma forma – aumenta o entendimento e organização;

- Existe um documento formal da Oracle que determinada esse padrão.

- http://www.devmedia.com.br/convencoes-de-codigo-java/23871

Convenções do Java

- Classes e Interfaces
  - class MinhaClasse
  - interface MinhaInterface
- Métodos e Variáveis
  - double valorSalarioMensal
  - calcularSalario()
- Constantes
  - DIAS_EXPIRACAO_SENHA
  - MAX_SIZE

Operadores e expressões

- Operador de atribuição simples
- A atribuição simples armazena o resultado da avaliação de uma expressão em uma variável e usa a sintaxe:
  - variavel=expressão;

  - Ex:
    - boolean resultado=false;
    - i=0;
    - y=a*x+b;
    - Byte m,n,p,q;
    - m=n=p=q=0; //equivale a m=(n=(p=(q=0)));

Operadores aritméticos

Operador	Significado	Exemplo
+	Adição	a+b
-	Subtração	a-b
*	Multiplicação	a*b
/	Divisão	a/b
%	Resto da divisão	a%b
++	Incremento	++a ou a++
--	Decremento	--a ou a__

Operadores aritméticos

```
public class P0206Aritmetica {
        public static void main (String args[]) {
                int a = 5, b = 2; // declara e inicia variáveis
                System.out.println("Valores: a = " + a + ", b = " + b);
                System.out.println(" -b = " + (-b)); // operações aritméticas
                System.out.println("a + b = " + (a + b));
                System.out.println("a - b = " + (a - b));
                System.out.println("a * b = " + (a * b));
                System.out.println("a / b = " + (a / b));
                System.out.println("(float) a / b = " + ((float)a / b));
                System.out.println("a % b = " + (a % b));
                System.out.println("a++ = " + (a++));
                System.out.println("  b = " + ( b));
                System.out.println("Valores: a = " + a + ", b = " + b);
        }
}
```

Operadores aritméticos

```
class ModDemo
{
  public static void main(String args[])
  {
    int iresult, irem;  double dresult, drem;  iresult = 10 / 3;
    irem = 10 %3;
    dresult = 10.0 / 3.0;
    drem = 10.0 %3.0;
      System.out.println("10 / 3 =" + iresult + " resto:" + irem);
      System.out.println("10.0 / 3.0=" + dresult + " resto:" + drem);
}
}
```

Saída:

```
10 / 3 =3 resto:1
10.0 / 3.0=3.3333333333333335 resto:1.0
```

Operadores de Incremento e Decremento

- x++; //adiciona 1 ao valor atual da variável
- x--; //subtrai 1 ao valor atual da variável
- int m=7;
- int n=7;
- int a=2*++m; //agora a é 16 , m=8
- int b=2*n++; //agora b é 14, n=8

Operadores Relacionais e Lógicos

Operador	Significado	Exemplo
==	Igual a	a==b
!=	Diferente de	a!=b
>	Maior	a>b
<	Menor	a<b
>=	Maior ou igual a	a>=b
<=	Menor ou igual a	a<=b

Operadores Relacionais e Lógicos

- **Operadores lógicos**

Operador	Significado	Exemplo
&	“E” Logico	a&b
|	“Ou” Logico	a|b
^	“Ou” Exclusivo	a^b
!	“Não”	!a
||	“Ou” de Curto Circuito	a||b
&&	“E” de Curto Circuito	a&&b

Ex: if ((cond1) & (cond2)) //Testa as duas sempre if((cond1)&&(cond2))
//Testa primeira se for falsa não avalia a segunda

p	q	p & q	p | q	p ^ q	!p
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

Expressões Relacionais e Lógicas

```
class RelLogOps
{ public static void main(String args[])
  {int i, j;
   boolean b1, b2;i = 10;
   j = 11;
   if(i < j)
     System.out.println("i < j");
   if(i <= j)
     System.out.println("i <= j");
   if(i != j)
     System.out.println("i != j");
   if(i == j)
     System.out.println("Este não executa");
   if(i >= j)
     System.out.println("Este não executa");
   if(i > j)
     System.out.println("Este não executa");
   b1 = true;  b2 = false;
   if(b1 & b2)
     System.out.println ("Este não executa");
   if(!(b1 & b2))
     System.out.println("!(b1 & b2) é verdade");
   if(b1 | b2)
     System.out.println("b1 | b2 é verdade");
   if(b1 ^ b2)
     System.out.println("b1 ^ b2 é verdade");
  }
}
```

Expressões Lógicas

- Operadores a nível de bit

~ Complemento

& AND

| OR

^ XOR

```
int a = 1234567890; // 0100 1001 1001 0110 0000 0010 1101 0010 em binár
int b = 987654321;  // 0011 1010 1101 1110 0110 1000 1011 0001 em binár
```

a & b (AND):

```
  0100 1001 1001 0110 0000 0010 1101 0010
& 0011 1010 1101 1110 0110 1000 1011 0001
= 0000 1000 1001 0110 0000 0000 1001 0000, equivalente à 144048272 dec:
```

a | b (OR):

```
  0100 1001 1001 0110 0000 0010 1101 0010
| 0011 1010 1101 1110 0110 1000 1011 0001
= 0111 1011 1101 1110 0110 1010 1111 0011, equivalente à 2078173939 de
```

Expressões Lógicas

- Operadores a nível de bit

~ Complemento

& AND

| OR

^ XOR

```
int a = 1234567890; // 0100 1001 1001 0110 0000 0010 1101 0010 em binái
int b = 987654321;  // 0011 1010 1101 1110 0110 1000 1011 0001 em binái
```

a ^ b (XOR):

```
  0100 1001 1001 0110 0000 0010 1101 0010
^ 0011 1010 1101 1110 0110 1000 1011 0001
= 0111 0011 0100 1000 0110 1010 0110 0011, equivalente à 1934125667 dec
```

~a (NOT):

```
~ 0100 1001 1001 0110 0000 0010 1101 0010
= 1011 0110 0110 1001 1111 1101 0010 1101, equivalente à -1234567891 de
```

Operador de Atribuição

Operador	Significado	Exemplo
+=	Adição e atribuição	a+=exp
-=	Subtração e atribuição	a-=exp
*=	Multiplicação e atribuição	a*=exp
/=	Divisão e atribuição	a/=exp
%=	Divisão inteira e atribuição	a%=exp
&=	E bitwise e atribuição	a&=exp
|=	Ou bitwise e atribuição	a|=exp

- var=expressão

Atribuições abreviadas

- var op=expressão;

x=x+10; equivalente a x+=10; x=x-100; equivale a x-=100;

- ONDE PESQUISAR
  - CAPÍTULO 4 - COREJAVA