Sumário

1. Introdução à Programação Orientada a Objetos

Introdução:

Computadores são ferramentas de uso comum hoje em dia. Eles podem automatizar tarefas e manipular valores. Para executar essa tarefa, computadores trabalham com modelos computacionais.

Um paradigma para definição desses modelos é fornecido pela Orientação a Objetos ou simplesmente OO. Nesse paradigma, os dados a serem processados e o processamento ou manipulação desses dados devem ser considerados em conjunto. Esse conjunto é denominado de modelo.

O que são modelos?

Modelos são representações simplificadas de objetos, pessoas, itens, tarefas, processos, conceitos, idéias, por exemplo.

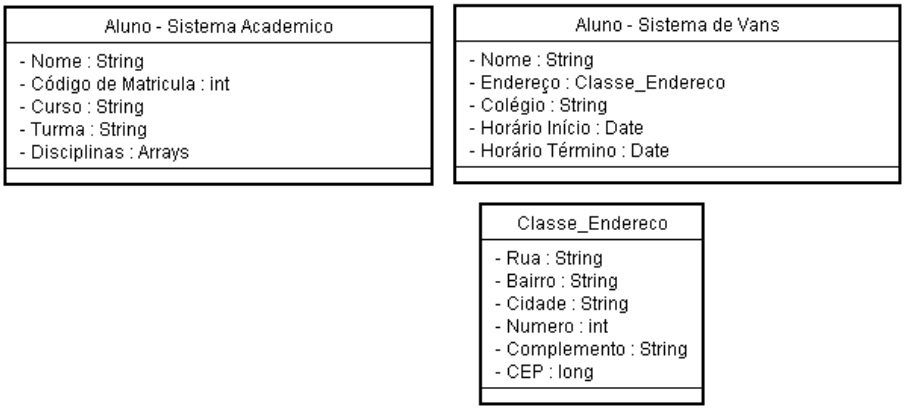
As simplificações inerentes aos modelos é, em muitos casos, necessária. Dependendo do contexto, algumas informações devem ser ocultas ou ignoradas.

Exemplo de Modelo:

Um modelo para a representação de um aluno para um sistema acadêmico, por exemplo, poderia ser formado com os seguintes dados: *Nome, Código de Matrícula, Curso, Turma, Disciplinas.*

Esse aluno poderia ser representado em um outro modelo se o objetivo fosse o desenvolvimento de um sistema de transporte escolar cuja função é levar e trazer o aluno da casa para o colégio. Nesse novo contexto, os dados poderiam ser, por exemplo: *Nome, Endereço, Colégio, horário de início da aula, horário de término da aula*.

Exemplo de Representação Gráfica do Modelo Aluno



O que é Programação Orientada a Objetos?

Programação Orientada a Objetos (POO) é um paradigma de programação de computadores onde se usam classes (modelos) e objetos (exemplos ou instâncias desses modelos) para o desenvolvimento de programas de computador.

Na POO, os dados pertencentes aos modelos são representados por tipos de dados nativos, ou seja, que são característicos da linguagem de programação, ou por outros modelos.

Por exemplo, o dado Nome poderia ser representado pelo tipo nativo String. Um dado do tipo String permite representar um conjunto de caracteres.

Por sua vez, Endereco poderia ser representado por um

modelo denominado por Classe_Endereco.

Encapsulamento:

Modelos podem conter dados para a representação das informações ou dados relativos e operações para manipulação desses dados.

Em algumas situações, esses dados não devem ser manipulados diretamente, mas somente através de processos. Como exemplo, o valor do salário de um funcionário poderia ser o resultado de um processamento "*[horas trabalhadas] x [valor hora]*".

Como analogia, pode-se considerar o televisor. Quando o usuário comum ligar o televisor apertando o botão "Ligar/Desligar" diversos mecanismos entram em ação.

Que mecanismos são esses e quais dados são importantes para esses mecanismos são informações sem importância para o usuário comum. O importante para ele é a possibilidade de assistir a programação desejada. Apesar dos dados e mecanismos não terem importância alguma, eles existem e estão escondidos ou encapsulados.

Estando encapsulados os dados, pode ser interessante prever recursos de modificação e leitura desses dados. Logo, deve-se criar uma operação para fazê-lo e ocultar o acesso direto a esses dados. A operação de acesso aos dados ocultos é denominado por **Métodos de Acesso**.

A capacidade de ocultar dados dentro de modelos, permitindo que somente operações especializadas ou dedicadas manipulem os dados ocultos chama-se **encapsulamento**.

Encapsulamento é um dos benefícios mais palpáveis da POO. Modelos que encapsulam os dados possibilitam a criação de programas com menos erros e mais clareza.

Exemplo 1:

A Figura 1 mostra um modelo Lampada usando uma variante do diagrama de classes da Linguagem Unificada de Modelagem (UML).

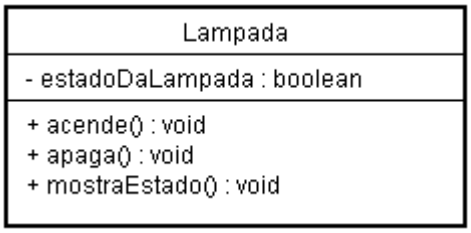


Figura 1 – Modelo Lampada

Neste diagrama o retângulo superior mostra o nome do modelo ou classe, o retângulo central mostra que dados definem o modelo e o retângulo inferior mostra que operações do modelo podem ser realizadas.

No modelo Lampada, alguns dados não estão representados, eles foram desprezados. São exemplos desses dados, consumo, cor e tamanho. A decisão de quais dados e operações devem pertencer a um modelo depende da abrangência e escopo do modelo.

O Exemplo 1 em Pseudocódigo:

```
Modelo Lampada
inicio do modelo
   dado estadoDaLampada;
   Operacao acende()
      Inicio
          estadoDaLampada = aceso;
      fim

   operacao apaga()
      inicio
          estadoDaLampada = apagado;
      fim

   operacao mostraEstado()
      inicio
          se (estadoDaLampada == aceso )
              imprime "A Lampada estah acesa";
          senão
              imprime "A Lampada estah apagada";
      fim
fim do modelo
```

Exemplo 2:

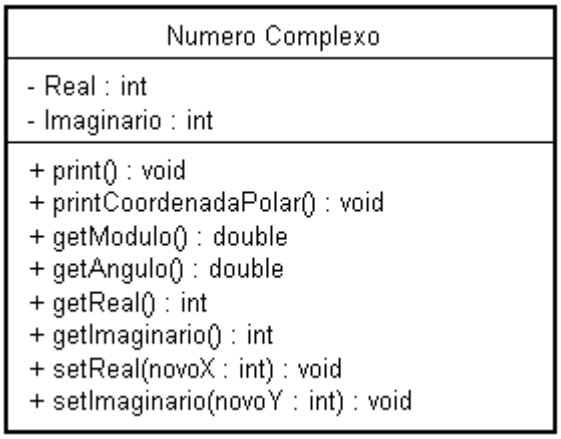


Figura 2 - Modelo Numero Complexo

O Exemplo 2 em Pseudocódigo:

```
Modelo NumeroComplexo
inicio do modelo
  dado Real, Imaginario;

operacao print()
  inicio
     escrever( " ( ");
     escrever ( Real );
     escrever( " ) ");
     escrever (" + j *( " );
     escrever (Imaginario);
     escrever (" ) ")
  fim

operacao printCoordenadaPolar()
  inicio
    dado Modulo, Angulo;
```

```
   Modulo = Real*Real + Imaginario*Imaginario ;
   Se (Real é igual a 0) entao Angulo = PI/2;
   Se (Imaginario é igual a 0) entao Angulo = 0;
   Se (Real não é igual a 0)
    e (Imaginario é igual a 0)
   entao  Angulo = Atan ( Imaginario / Real );
  Fim

operacao getModulo()
  inicio
    retorne (Real*Real + Imaginario*Imaginario);
  fim

operacao getAngulo()
   inicio
      retorne Atan ( Imaginario / Real );
   fim

operacao setReal(novoX)
   inicio
      se (novoX não é igual a Real)
      entao Real = novoX;
   fim
operacao setImaginario(novoY)
    inicio
      se (novoY não é igual a Imaginario)
      entao Imaginario = novoY;
    fim
fim do modelo
```

Exemplo 3:

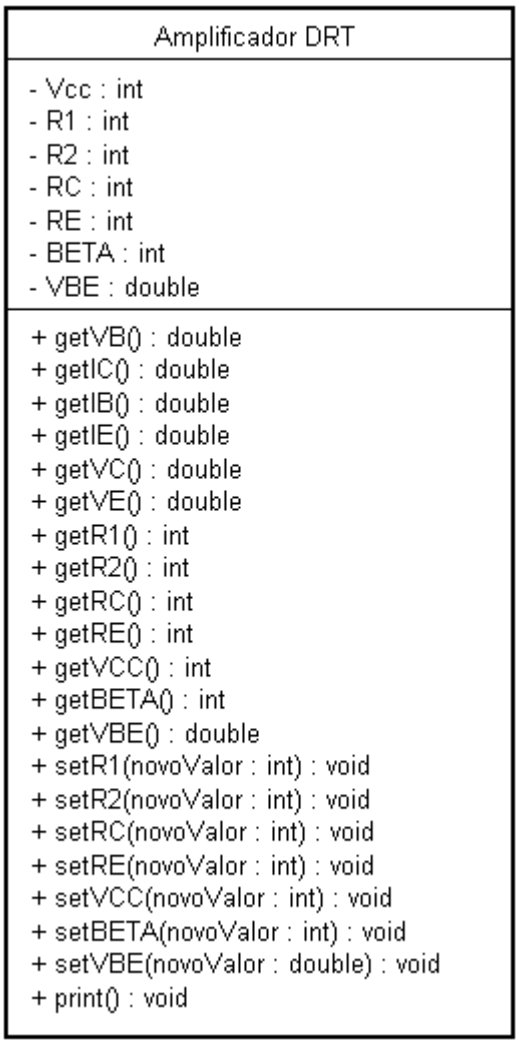


Figura 3 - Modelo AmplificadorDRT

O Exemplo 3 em Pseudocódigo:

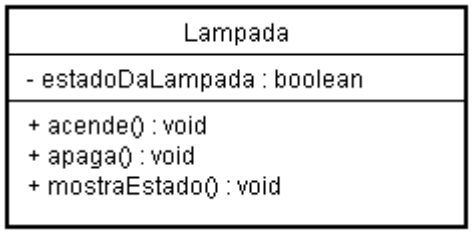
```
Modelo AmplificadorDRT
inicio do modelo
   dado Vcc, R1, R2, RC, RE, BETA, VBE;

   operacao getVB()
   inicio
      dado VB = -1;
      if ( (R1+R2) não é igual a 0 )
      entao  VB = Vcc*R2/(R1+R2);
      senao escrever("ERRO de Valor);
      retorne VB;
   fim

   operacao getIC()
   inicio
     dado valor=-1;
     if ( RE não é igual a 0)
     entao valor = getIE();
     senao valor = BETA * getIB();
     valor = valor*(BETA)/(BETA+1);
     retorne valor;
   fim
```

```
operacao getIE()
inicio
  dado valor=-1;
  if (RE não é igual a zero)
     valor = getVE() / RE;
  retorne valor;
fim

operacao getIB()
inicio
  dado valor = -1;
  if (RC é igual a 0)
  entao  valor  =  (getVB()  -  getVBE())  /
            (R1*R2/(R1+R2));
  senao
  if (BETA nãp é igual a 0)
      valor = getIC()/BETA;
   retorne valor;
fim

operacao getVC()
inicio
   dado valor;
    valor = VCC - RC*getIC();
   retorne valor;
fim

operacao getVE()
inicio
    dado valor;
    valor = getVB() - getVBE();
    retorne valor;
fim

operacao getR1()
inicio
   retorne R1;
fim

operacao getR2()
inicio
   retorne R2;
fim

operacao getRC()
inicio
   retorne RC;
fim

/* algumas instruções foram omitidas */

fim do modelo
```

Classes e Objetos:	Programadores que utilizam o paradigma de POO criam e usam *objetos* a partir de *classes*. *Classes* estão relacionadas diretamente com modelos. Classes são estruturas que definem os dados que devem ser representados e as operações que devem ser efetuadas com esses dados. Objetos são exemplos dessas classes. Uma analogia pode ser realizada nesse instante. Pode-se considerar que a receita para fazer um bolo é uma *classe*. Nela estão representados os ingredientes ou dados (ovo, farinha, etc...) e os processos necessários (quebrar o ovo, separar a gema da clara, etc...). Quando, a partir da receita, um bolo é feito, esse bolo é o *objeto*.
Softwares Recomendados:	• Interface de Edição: JCreator LE – www.jcreator.com ou NetBeans – www.netbeans.org ou Eclipse – www.eclipse.org • Compilador e Máquina de Execução: Java SDK – www.sun.com
Referência:	Título:Introdução à Programação Orientada a Objetos usando Java. Ano: 2003 Autor: Rafael Santos Editora: Campus ISBN: 85-352-1206-X
Literatura de Apoio:	Título: Core Java, Volume 1 – Fundamentos Ano: 2001 Autor: Cay S. Hortsmann e Gary Cornell Editora: Makron Books ISBN: 85-346-1225-0

2. Qual a relação entre modelo e linguagem Java?

Modelos são implementados em Java através de classes. Por exemplo:

<table>
<tr><th>Modelo</th><th>Código Java</th></tr>
<tr><td> 
</td><td rowspan="2">

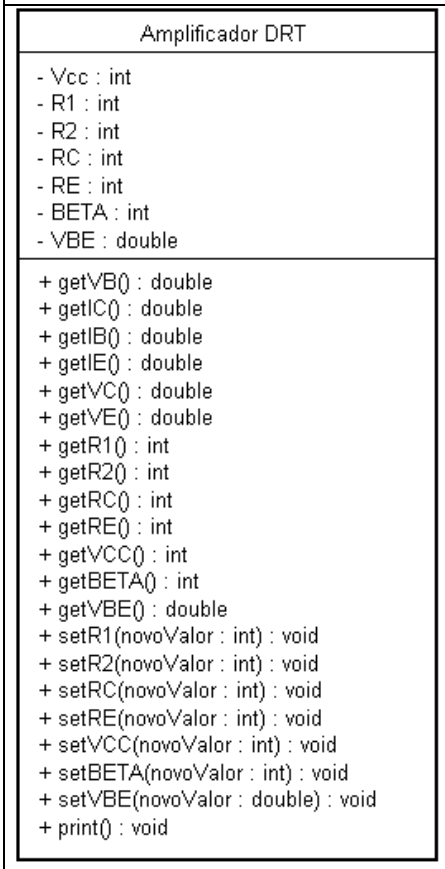
```
// arquivo detalhado Lampada.java
class Lampada
{
  private boolean
       estadoDaLampada=false;

  public void acende()
  { estadoDaLampada = true; }

  public void apaga()
  { estadoDaLampada = false; }

  public void mostraEstado()
  { if (estadoDaLampada == true)
     System.out.println(
     "A lampada estah acesa.");
    else
     System.out.println(
      "A lampada estah apagada.");
  }
}
```

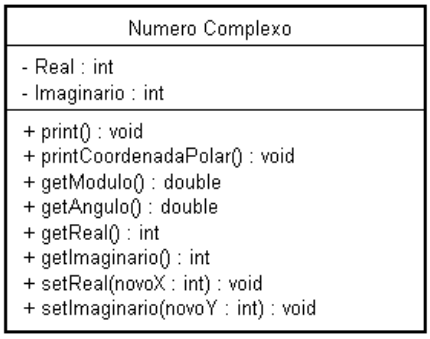
</td></tr>
<tr><td>

```
Modelo Lampada
Inicio do modelo
 dado estadoDaLampada;

 operacao acende()
 inicio
  estadoDaLampada = aceso;
 fim

 operacao apaga()
 inicio
  estadoDaLampada = apagado;
 fim

 operacao mostraEstado()
 inicio
  se   (estadoDaLampada   ==
aceso )
    imprime "A Lampada estah
acesa";
  senão
    imprime "A Lampada estah
apagada";
 fim
fim do modelo
```

</td></tr>
</table>

Diagrama UML	Esqueleto de Código Java

	`public class AmplificadorDRT {`
	`private int Vcc;`
	`private int R1;`
	`private int R2;`
	`private int RC;`
	`private int RE;`
	`private int BETA;`
	`private double VBE;`
	`public double getVB() {return 0;}`
	`public double getIC() {return 0;}`
	`public double getIB() {return 0;}`
	`public double getIE() {return 0;}`
	`public double getVC() {return 0; }`
	`public double getVE() {return 0;}`
	`public int getR1() {return 0;}`
	`public int getR2() {return 0;}`
	`public int getRC() {return 0;}`
	`public int getRE() {return 0;}`
	`public int getVCC() {return 0;}`
	`public int getBETA() {return 0;}`
	`public double getVBE() {return 0;}`
	`public void setR1(int novoValor) {}`
	`public void setR2(int novoValor) {}`
	`public void setRC(int novoValor) {}`
	`public void setRE(int novoValor) {}`
	`public void setVCC(int novoValor) {}`
	`public void setBETA(int novoValor) {}`
	`public void setVBE(double novoValor) {}`
	`public void print() { }`
	`}`

	`public class NumeroComplexo {`
	`private int Real;`
	`private int Imaginario;`
	`public void print() { }`
	`public void printCoordenadaPolar() {}`
	`public double getModulo() { return 0; }`
	`public double getAngulo() { return 0;}`
	`public int getReal() { return 0;}`
	`public int getImaginario() { return 0; }`
	`public void setReal(int novoX) { }`
	`public void setImaginario(int novoY) { }`
	`}`

3. Criação de Classes em Java

Sintaxe Básica:

- Uma classe em Java é sempre declarada com a palavra-chave ***class*** seguida do nome da classe. O nome da classe não pode conter espaços e deve sempre ser iniciado por uma letra. Para nomes de classes, métodos e campos em Java, o caracter sublinhado (_) e o sinal (*$*) também são considerados letras.
- O nome da classe não deve conter acentos e pode conter números, contanto que estes apareçam depois de uma letra.
- Nomes de Classes não podem ser exatamente iguais às palavras reservadas em Java.
- Caracteres maiúsculos e minúsculos são diferenciados em Java.
- O conteúdo das classes é delimitado pelas chaves "{" e "}". Todos os campos e métodos também devem estar entre esses caracteres.
- Uma classe declarada como pública necessita obrigatoriamente ser salva em um arquivo .java e o nome do arquivo necessita ser o mesmo da classe.

Palavras Reservadas:

`abstract, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, extends, false, final, finally, float, for, goto, if, implements, import, instanceof, it, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transiente, true, try, void, volatile, while.`

Tipos Básicos em Java:

Tipo	Faixa de valores
`boolean`	`true` ou `false`
`char`	0 a 65535
`byte`	-128 a 127
`short`	-32768 a 32767
`int`	-2147483648 a 2147483647
`long`	-9223372036854775808 a 9223372036854775807
`float`	1.40129846432481707 e – 45 a 3.40282346638528860 e + 38
`double`	4.9406545841246544 e – 324 a 1.7976931348623157 e + 308
`String`	Tamanho limitado à memória disponível

Uma classe é uma aplicação ?

A diferença entre uma classe e uma aplicação consiste na existência de um método de execução:

```
public static void main(String [] a)
{  comandos ; }
```

Exemplo de aplicação:

```
// arquivo TesteOlaMundo.java
public class TesteOlaMundo
{
  public static void main(String [] a)
  {
    System.out.println("Ola Mundo !");
  }
}
```

Exemplo de não aplicação:

```
// arquivo OlaMundo.java
public class OlaMundo
{
  public void main()
  {
    System.out.println("Ola Mundo !");
  }
}
```

Diferença entre classe e aplicação

Os arquivos TesteOlaMundo.java e OlaMundo.java definem duas classes distintas.

A classe TesteOlaMundo é uma aplicação pois ela possui o método "`public static void main(String [] a)`".

A classe OlaMundo não é uma aplicação pois ela não possui o método "`public static void main(String [] a)`". Essa classe possui um médoto parecido, o método "`public void main()`".

A diferença entre "`public static void main(String [] a)`" e "`public void main()`" é a presença das palavras "`static`" e "`String [] a`".

Como criar um objeto de dentro de uma aplicação?

```
/*1.*/  // arquivo testeLamp.java
/*2.*/  class Lampada
/*3.*/  {
/*4.*/  private boolean estadoDaLampada=false;

/*5.*/  public void acende()
/*6.*/  { estadoDaLampada = true; }

/*7.*/  public void apaga()
/*8.*/  { estadoDaLampada = false; }

/*9.*/  public void mostraEstado()
/*10.*/ { if (estadoDaLampada == true)
/*11.*/ System.out.println("A lampada estah acesa.");
/*12.*/ else
```

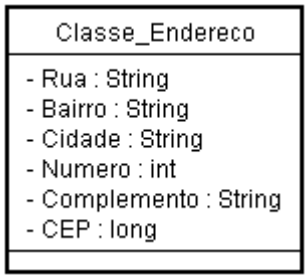
```
/*13.*/ System.out.println("A lamada estah apagada.");
/*14.*/ }
/*15.*/ }
/*16.*/ public class testeLamp
/*17.*/ {
/*18.*/ public static void main(String[] a)
/*19.*/ {
/*20.*/ Lampada lamp1 = new Lampada();
/*21.*/ lamp1.acende();
/*22.*/ lamp1.mostraEstado();
/*23.*/ lamp1.apaga();
/*24.*/ lamp1.mostraEstado();
/*25.*/ }
/*26.*/ }
```

A palavra-chave `new`

A criação e o uso de instâncias de classes em métodos como o "`public static void main(String[] a)`", por exemplo, mostrado no exemplo anterior são feitos através da palavra-chave **`new`**.

Recomendações para o projeto de classes:

1. Sempre mantenha os dados privados. Desenvolva metodos de acesso (get) e alteracao (set).
2. Sempre inicialize os dados
3. Nao use tipos basicos em demasia numa classe. A idéia é substitui-los por uma nova classe. Por exemplo, os dados relativos ao endereço poderiam ser substituidos por uma única classe Classe_Endereco.



Exercícios:

1) Criar um modelo para representar um triângulo. O modelo deverá armazenas os três lados, verificar se os dados dos três lados formam um triângulo e calcular o perímetro do triângulo.

2) Implementar esse modelo em Java.

3) Criar um modelo para representar as coordenadas de um ponto em um sistema bidimensional. Esse modelo deverá incluir um método para mover o ponto, ou seja, modificar as suas coordenadas.

4) Implementar o modelo do exercício 3 em Java.

5) Implementar em Java o modelo RegistroAcademico.

6) Fazendo uso do modelo desenvolvido no exercício 3, especifique um modelo que represente uma reta.

7) Implemente o modelo do exercício 6 em Java.

4. Aplicações em Java

Em Java, podemos criar em uma classe um método especial que será considerado o ponto de entrada do programa. A presença desse método na classe fará com que a classe se torne executável. Dentro desse método poderemos ter a criação e manipulação de dados e instâncias de classes.

Esse método é definido pela seguinte assinatura:

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

Exemplo de definição de classe:

```
                                                    // arquivo Ponto2D.java
/*1.*/  public class Ponto2D {
/*2.*/  private double x,y;
/*3.*/
/*4.*/   public void inicializaPonto2D(double _x,double _y)
/*5.*/   { x = _x;
/*6.*/    y = _y;
/*7.*/   }

/*8.*/   public boolean ehIgual(Ponto2D outroPonto2D)
/*9.*/   {
/*10.*/   if ((x == outroPonto2D.x) &&
/*11.*/       (y == outroPonto2D.y))
/*12.*/        return true;
/*13.*/   else
/*14.*/        return false;
/*15.*/  }
/*16.*/  public Ponto2D origem()
/*17.*/  {
/*18.*/   Ponto2D temporario = new Ponto2D();
/*19.*/   temporario.inicializaPonto2D(0,0);
/*20.*/   return temporario;
/*21.*/  }
/*22.*/  public Ponto2D clona()
/*23.*/  {
/*24.*/   Ponto2D temporario = new Ponto2D();
/*25.*/   temporario.inicializaPonto2D(x,y);
/*26.*/   return temporario;
/*27.*/  }

/*28.*/  public String toString()
/*29.*/  {
/*30.*/   String resultado = "("+x+","+y+")";
```

```
/*31.*/   return resultado;
/*32.*/  }

/*33.*/  public double distancia(Ponto2D ext)
/*34.*/  { double saida = -1.0;
/*35.*/    saida = Math.sqrt( (ext.x - x)*(ext.x - x)
/*36.*/                + (ext.y - y)*(ext.y - y) );
/*37.*/  return saida;
/*38.*/  }
/*39.*/ }
```

Quadro 1 Exemplo da Classe Ponto2D

Exemplo de Aplicação:

```
                                                    // arquivo Ponto2D.java
/*1.*/   public class Ponto2D {
/*2.*/   private double x,y;
/*3.*/   public void inicializaPonto2D(double _x,double _y)
/*4.*/   { x = _x;
/*5.*/     y = _y; }

/*6.*/   public boolean ehIgual(Ponto2D outroPonto2D)
/*7.*/   { if ((x == outroPonto2D.x) &&
/*8.*/         (y == outroPonto2D.y))
/*9.*/      return true;
/*10.*/    else
/*11.*/     return false; }

/*12.*/  public Ponto2D origem()
/*13.*/  { Ponto2D temporario = new Ponto2D();
/*14.*/    temporario.inicializaPonto2D(0,0);
/*15.*/    return temporario;  }

/*16.*/  public Ponto2D clona()
/*17.*/  { Ponto2D temporario = new Ponto2D();
/*18.*/    temporario.inicializaPonto2D(x,y);
/*19.*/    return temporario; }

/*20.*/   public String toString()
/*21.*/   { String resultado = "("+x+","+y+")";
/*22.*/    return resultado; }

/*23.*/  public double distancia(Ponto2D ext)
/*24.*/  { double saida = -1.0;
/*25.*/    saida = Math.sqrt( (ext.x - x)*(ext.x - x)
/*26.*/              + (ext.y - y)*(ext.y - y) );
```

```
/*27.*/   return saida;
/*28.*/  }

/*29.*/ public static void main(String[] argumentos)
/*30.*/ {  Ponto2D p1;
/*31.*/    Ponto2D p2,p3,p4;
/*32.*/    p1 = new Ponto2D();
/*33.*/    p2 = new Ponto2D();
/*34.*/    p1.inicializaPonto2D(-1.34,9.17);
/*35.*/    System.out.println("As coordenadas de P1 sao " +
                              p1);
/*36.*/    System.out.println("As coordenadas de P2 sao " +
                              p2);
/*37.*/     P3 = p1.clona();
/*38.*/     p4 = p1.origem();
/*39.*/     System.out.println("As coordenadas de P3 sao "
                              + p3);
/*40.*/     System.out.println("As coordenadas de P4 sao "
                               + p4);
/*41.*/     System.out.println("P1 estah na mesma posicao "
                            +" de P2 ? " + p1.ehIgual(p2));
/*42.*/     System.out.println("P1 estah na mesma posicao "
                             +" de P3 ? " + p1.ehIgual(p3));
/*43.*/     System.out.println("P1 estah na mesma posicao "
                             +" de P4? " +p1.ehIgual(p4));
/*44.*/     System.out.println(
/*45.*/        "P2 estah na mesma posicao de P4 ? "
/*46.*/              +p2.ehIgual(p4));
/*47.*/     System.out.println("Novo Ponto:"
/*48.*/             + new Ponto2D());
/*49.*/     System.out.println(
/*50.*/        "A distancia entre P1 e P2 eh "
/*51.*/        + p1.distancia(p2));
/*52.*/ }
/*53.*/  }
```

Quadro 2 - Aplicação Ponto 2D

*** Observe que a principal diferença entre os exemplos dos Quadros 1 e 2 é a existência do método

```
public static void main(String[] argumentos)
{/* aqui vai o código que define a aplicação*/ }
```

****

5. Estruturas Fundamentais em Java

Comentários:

```
// duas barras indica que a partir de seu ponto de inserção, o
// resto da linha será comentário.
/*  define um bloco de comentários */
```

Tipos de dados:

boolean	Valores : true / false
int	Intervalo de -2147483648 a 2147483647
short	Intervalo de -32768 a 32767
long	-9223372036854775808 a 9223372036854775807
byte	-128 1 127
float	+ - 3.40282347 e + 38
double	+ - 1.7976931348623157 e 308
char	65536 especiais: **\b** backsapce **\t** tabulação **\n** nova linha **\r** retorno do carro \" aspas \' apóstrofe \\ barra invertida.

Conversões entre tipos:

```
                                        // arquivo testeConversao.java
/*1.*/  class testeConversao
/*2.*/  { public static void main(String [] a)
/*3.*/    {  double x1 = 3.47;
/*4.*/       char letraA = 'a';
/*5.*/       char letraB = 'b';
/*6.*/       char letraC = 'c';
/*7.*/       int y1 = (int)x1;
/*8.*/       int y2 = (int)letraA;
/*9.*/       System.out.println("y1="+ y1
/*10.*/                        + " y2=" + y2);
/*11.*/    System.out.println( (int)letraB );
/*12.*/    System.out.println( (int)letraC );
/*13.*/  }
/*14.*/   }
```

Operadores Lógicos:

==	`If ( 3 == 4 )` // testa se 3 é igual a 4
!=	`If ( letra != 'c')` // testa se letra é diferente de c.
>	`If ( 3 < 4 )` // testa se 3 é menor que 4
<	`If ( 3 > 4 )` // testa se 3 é maior que 4
>=	`If ( x >= 4 )` // testa se x é maior ou igual a 4
<=	`if ( x <= 4 )` // testa se x é menor ou igual a 4
&&	`if (( x < 4 )&& ( x > 1) )` // testa se x é menor que 4 e maior que 1

||	if ((x < 4)|| (x > 1)) // testa se x é menor que 4 ou maior que 1

6 – Arrays

A estrutura de dados do tipo Array permite armazenar uma coleção de objetos ou elementos do mesmo tipo.

Por exemplo, um Array de inteiros permite armazenar um conjunto de números inteiros. Um Array de Ponto2D permite armazenar um conjunto de objetos da classe Ponto2D.

6.1 Criação

Um Array precisa ser ser criado explicitamente conforme o exemplo a seguir:

```
int [ ] numeros = new int[10];
for (int i = 0; i < 10; i++) numeros[i]=i;
System.out.println( numeros[9] );
```

Nesse exemplo, um array denominado por **numeros** é criado com a palavra reservada **new**. Em seguida, esse Array é preenchido com números inteiros através da instrução **for (; ;).**

A seguinte instrução imprimirá o número 9 na saída:

```
System.out.println( numeros[9] );
```

Observação:

A palavra reservada **new** já foi utilizada anteriormente para criação de instâncias de classe, como por exemplo **Ponto2D p1 = new Ponto2D()**.

O que distingue o seu uso é a presença dos "(" e ")" ou das "[" "]".

A palavra chave `new` *sendo utilizada com "**[**" e "**]**" define a criação de um array.*

*Quando usado com "**(**" e "**)**" é instância de classe.*

Um array também pode ser criado de maneira mais rápida quando é conhecido o conjunto de elementos que ele armazenará. Por exemplo:

```
char[ ] vogais = {'a', 'e', 'i', 'o', 'u'};
```

Nesse exemplo, o array "vogais " é criado a partir do conjunto de cinco elementos fornecidos na mesma instrução. Além de criar o array, a instrução preenche o array com valores.

O posicionamento dos "[" e "]" pode ser feito antes ou após da definição da variável. Por exemplo, os dois códigos abaixo são aceitos pelo compilador Java:

```
int [ ] numeros = new int[10];
int numbers [ ] = new int[10];
```

6.2 Arrays de Objetos.

Um conjunto de objetos pode ser armazenado em um Array. Por exemplo:

```
Lampada[ ] pinheirinho = new Lampada[8];
```

Nesse exemplo, um array denominado por "pinheirinho" deverá armazenar oito (8) objetos do tipo Lâmpada.

CUIDADO: Nesse ponto, que ainda não ocorreu a geração de instâncias de cada objeto da classe Lâmpada. A seguinte instrução é necessária e executa essa tarefa.

```
for(int i = 0; i < 8; i++) pinheirinho[i] = new Lampada();
```

Nesse exemplo a criação de objetos Lampada é feita explicitamente.

6.3 Array como argumento.

Um Array pode ser utilizado como um argumento para uma função. Ou seja, a função não receberá apenas um único elemento mas uma coleção. Quando isso ocorre, dentro da função, o programador não necessita prever quantos elementos estão contidos nessa coleção.

Diferentemente de outras linguagens de programação, como o C, um array em java é uma estrutura de informações. Dessa estrutura, uma das principais informações é o tamanho ou **length** do array. No exemplo a seguir, a função imprimeImpares(int [ ]) recebe um array de tamanho qualquer. O tamanho do array recebido é obtido pela chamada do atributo **length** na linha /*11*/ do código.

```
/////arquivo demoArrays.java
/*1*/public class demoArrays
/*2*/{

/*3*/  public static void main(String[] a)
/*4*/  {
/*5*/      int [] numeros = {1,2,3,4,5,6,7,8,9,10};
/*6*/      imprimeImpares(numeros);
/*7*/  }

/*8*/  public static void imprimeImpares(int [] lista)
/*9*/  {
/*10*/     double resto; int quantidade=0;
/*11*/     for (int i = 0; i < lista.length; i++)
/*12*/     {   resto = lista[i] % 2.0;
/*13*/         if ( resto != 0.0)
```

```
/*14*/         {System.out.println(lista[i]); quantidade++;}
/*15*/     }
/*16*/     System.out.print("\n *** Recebi um Array com "
/*17*/                      + lista.length + " elementos.");
/*18*/     System.out.println("\t Apenas " + quantidade
/*19*/                      + " elementos são impares.\n");
/*20*/   }
/*21*/}
```

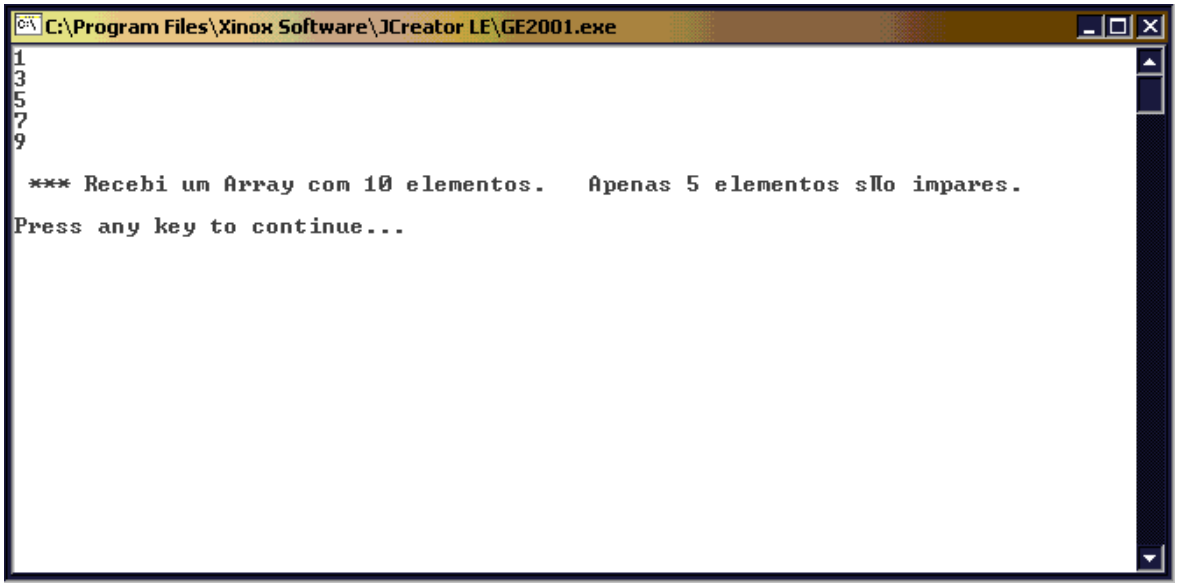


Figura 4-Exemplo de Execução do programa demoArrays.java

6.4 Array como valor de retorno

O valor de retorno de uma função pode ser um Array. Nesse caso, não há a necessidade de especificar explicitamente no código qual é o tamanho desse array.

O programa a seguit ilustrar esse conceito.

```
/////arquivo demoArrays.java
/*1*/public class inverteArray
/*2*/{
/*3*/  public static void main(String[] a)
/*4*/  {
/*5*/   int [] numeros = {1,2,3,4,5,6,7,8,9,10};
/*6*/   int [] numerosInv = inverteOrdem(numeros);
/*6*/   imprimeArray(numeros);
/*7*/   System.out.println("\n\n >Numeros Invertidos:");
/*8*/                  imprimeArray(numerosInv);
/*7*/  }
/*8*/  public static void imprimeArray(int [] lista)
/*9*/  {
/*10*/   System.out.println("\n**** Lista com "
```

```
/*11*/            + lista.length + " elementos:");

/*12*/    for (int i = 0; i < lista.length; i++)
/*13*/      System.out.println(lista[i]);
/*14*/
/*15*/   }
/*16*/   public static int[ ] inverteOrdem(int [ ] lista)
/*17*/   {
/*18*/     int[] saida = new int[lista.length];
/*19*/     for (int i = 0; i < lista.length; i++)
/*20*/        saida[saida.length-i-1] = lista[i];
/*21*/     return saida;
/*22*/   }
/*23*/}
```

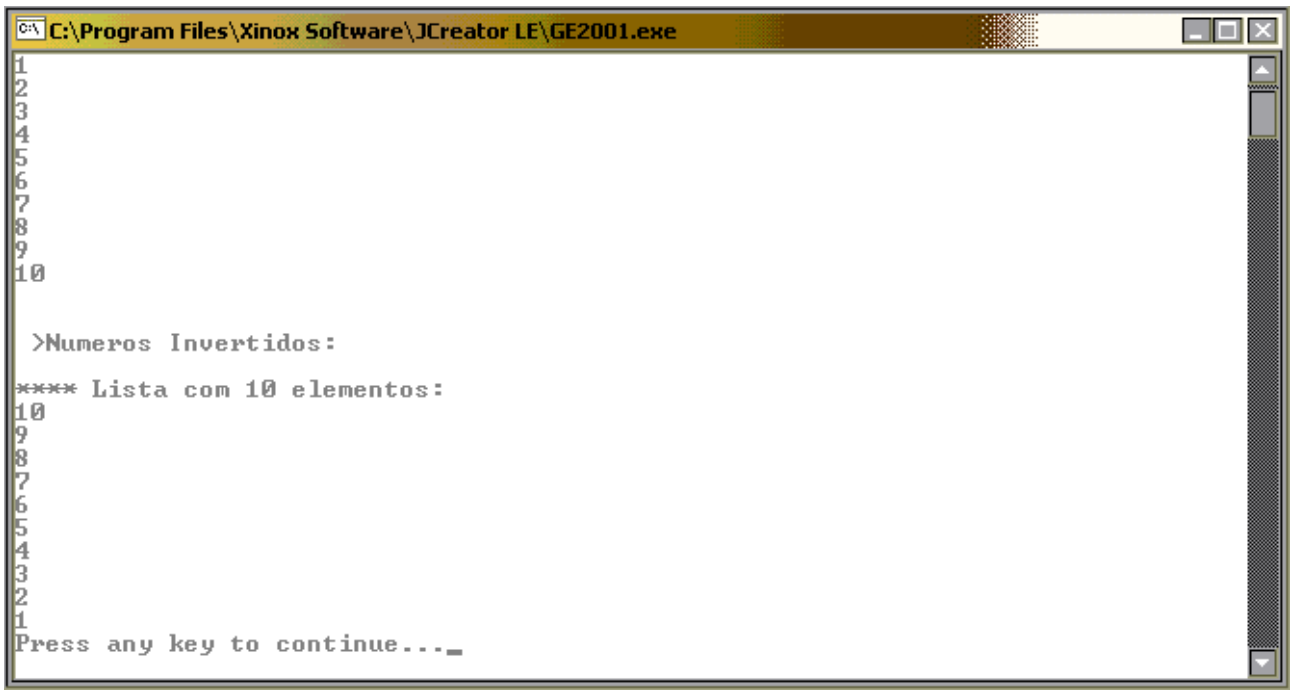


Figura 5-Exemplo de Execução do programa inverteArray.java

Observe no código anterior que a função "**`/*16*/ public static int[ ] inverteOrdem(int [ ] lista)`**" recebe como um argumento um array de inteiros denominado `lista` e essa função retorna um array de inteiros.

6.5 Exercícios.

1. Criar uma função que imprima um array de objetos Ponto2D.
2. Criar uma função que receba uma array de objetos Triangulo e verifique quantos desses elementos são equiláteros.

7 – A classe java.util.Vector

A classe java.util.Vector armazena elementos em uma lista. Essa classe armazena qualquer tipo de elemento, sem a necessidade prévia de determinar que tipo de elemento será armazenado.

Por exemplo, o programa do quadro a seguir, define duas classes ClasseTipo1 e ClassetTipo2 nas linhas /*2*/ e /*3*/. Nas linhas /*9*/ e /*10*/ duas instâncias dessas classes são criadas, `e1` e `e2`, respectivamente.

Observe que essas classes são adicionadas ao vetor nas linhas /*13*/ e /*14*/, sem a necessidade prévia de alguma configuração específica. Nas linhas /*29*/ e /*32*/ os elementos armazenados no vetor são copiados para fora do vetor. Nesse instante, verifica-se de qual classe o objeto é instância.

```
/*1.*/  // aplicação exemploVector.java

/*2.*/  class ClasseTipo1 { /* uma classe que nao representa nada.*/}

/*3.*/  class ClasseTipo2 { /* outra classe que nao representa nada. */}

/*4.*/  public class exemploVector
/*5.*/  {  public static void main(String[] a)
/*6.*/     {
/*7.*/       java.util.Vector  vetor = new java.util.Vector();

/*8.*/       // instância de objetos que serão armazenados no vetor.
/*9.*/       ClasseTipo1 e1 = new ClasseTipo1();
/*10.*/      ClasseTipo2 e2 = new ClasseTipo2();
/*11.*/      String texto = "Ola Mundo!";

/*12.*/      // instrução para armazenar os objetos no final da lista.
/*13.*/      vetor.addElement(e1);
/*14.*/      vetor.addElement(e2);
/*15.*/      vetor.addElement(1);
/*16.*/      /* instrução para armazenar o objeto em uma posição
   específica da lista*/
/*17.*/      vetor.insertElementAt(texto,1);
/*18.*/      imprimeVetor(vetor);
/*19.*/      // remove o elemento da posição 0.
/*20.*/      vetor.remove(0);
/*21.*/      imprimeVetor(vetor);
/*22.*/ }
```

```
/*23.*/ public static void imprimeVetor(java.util.Vector lista)
/*24.*/ { if (lista.isEmpty()) return;
/*25.*/   System.out.print("\n");
/*26.*/   for (int j = 0; j < 50; j++) System.out.print("-");
/*27.*/   System.out.print("\n");
/*28.*/   for (int i = 0; i < lista.size(); i++)
/*29.*/   {  if (lista.elementAt(i) instanceof ClasseTipo1)
/*30.*/     {System.out.println("-" + (i+1)
                        + "> Instancia da ClasseTipo1");}
/*31.*/     else
/*32.*/      if (lista.elementAt(i) instanceof ClasseTipo2)
/*33.*/       {System.out.println("-" + (i+1)
                           + "> Instancia da ClasseTipo2");}
/*34.*/      else
/*35.*/       System.out.println("-" + (i+1) + "> "
                         + lista.elementAt(i));
/*36.*/  }}}
```

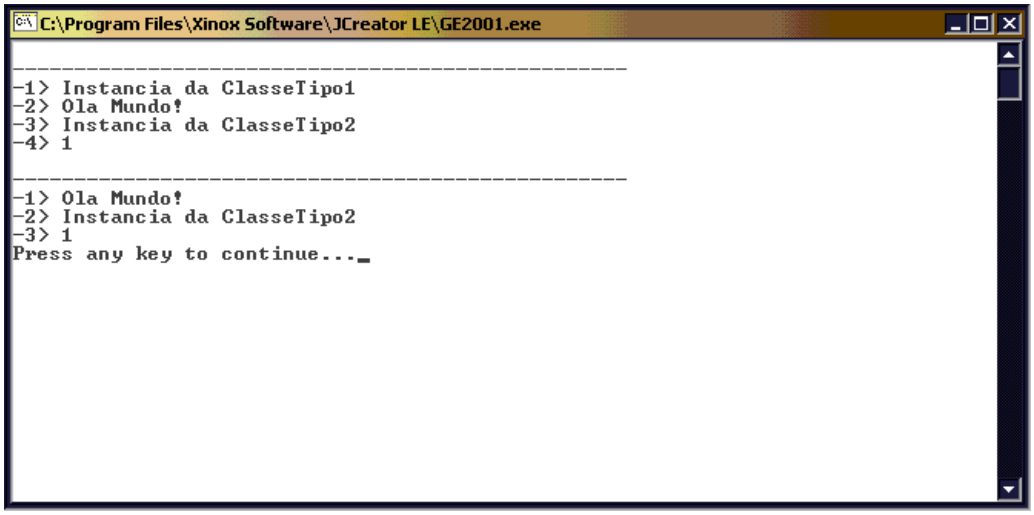


Figura 6- Exemplo de Execução do programa exemploVector.java

8 - Laços

Laço indeterminado:

```
/*1.*/
/*2.*/ class contador
/*3.*/ {
/*4.*/ public static void main(String [] a)
/*5.*/ {
/*6.*/  int x = 10;
/*7.*/  while (x > 0)
/*8.*/ {
/*9.*/   System.out.println(x);
/*10.*/  x-- ;
/*11.*/ }
/*12.*/ }
/*13.*/ }
```

Laço indeterminado 2:

```
/*1.*/ class contador2
/*2.*/ {
/*3.*/ public static void main(String [] a)
/*4.*/ {
/*5.*/ int x = 10;
/*6.*/ do
/*7.*/ {
/*8.*/     System.out.println(x);
/*9.*/     x-- ;
/*10.*/   }while (x > 0);
/*11.*/ }
/*12.*/ }
```

Laço determinado:

```
/*1.*/ class contador3
/*2.*/ {
/*3.*/ public static void main(String [] a)
/*4.*/ {
/*5.*/    for (int x = 10; x >0; x--)
/*6.*/          System.out.println(x);
/*7.*/ }
/*8.*/ }
```

Interrupção de laço:

- **`break;`** // interrompe o laço
- **`continue;`** // realiza uma nova iteração
- **`return;`** // abandona o método em execução.
- **`System.exit(0)`**; // encerra a aplicação com retorno 0 (zero).

Exemplo de interrupção continue:

```
/*1.*/ class divisao
/*2.*/ {  public static void main(String[] a)
/*3.*/ {
/*4.*/   for (int i = 1; i < 100; i++)
/*5.*/   {   if ( (i%3) == 0) continue;
/*6.*/       System.out.println(i);
/*7.*/   }
/*8.*/   }}
```

Exemplo de interrupção `break`:

```
/*1.*/  class divisao2
/*2.*/  {  public static void main(String[] a)
/*3.*/  {   for (int i = 1; i < 100; i++)
/*4.*/  {       if ( (i%7) == 0) break;
/*5.*/      System.out.println(i); }
/*6.*/      System.out.println("fim");
/*7.*/  } }
```

Exemplo de interrupção `return`:

```
/*1.*/  class divisao3
/*2.*/  {  public static void main(String[] a)
/*3.*/     {  for (int i = 1; i < 100; i++)
/*4.*/       { if ( (i%7) == 0) return;
/*5.*/         System.out.println(i);
/*6.*/       }
/*7.*/       System.out.println("fim");
/*8.*/     }
/*9.*/  }
```

9 - A Classe String

String:

Uma String é uma sequência de caracteres. Diferentemente dos outros tipos de dados, uma String é uma classe.

Como criar uma String:

```
String mensagem = "";
String mensagem = new String();
```

Como concatenar uma String?

```
/*1.*/  // arquivo concatenaString.java
/*2.*/  class concatenaString
/*3.*/  {   public static void main(String [] a)
/*4.*/  {   String p1 = "ola";
/*5.*/      String p2 = "mundo";
/*6.*/      String p3 = p1 + p2;
/*7.*/      String p4 = p1 + "  " + p2;
/*8.*/      System.out.println("1-" + p3);
/*9.*/      System.out.println("2-" + p4);
/*10.*/ }}
```

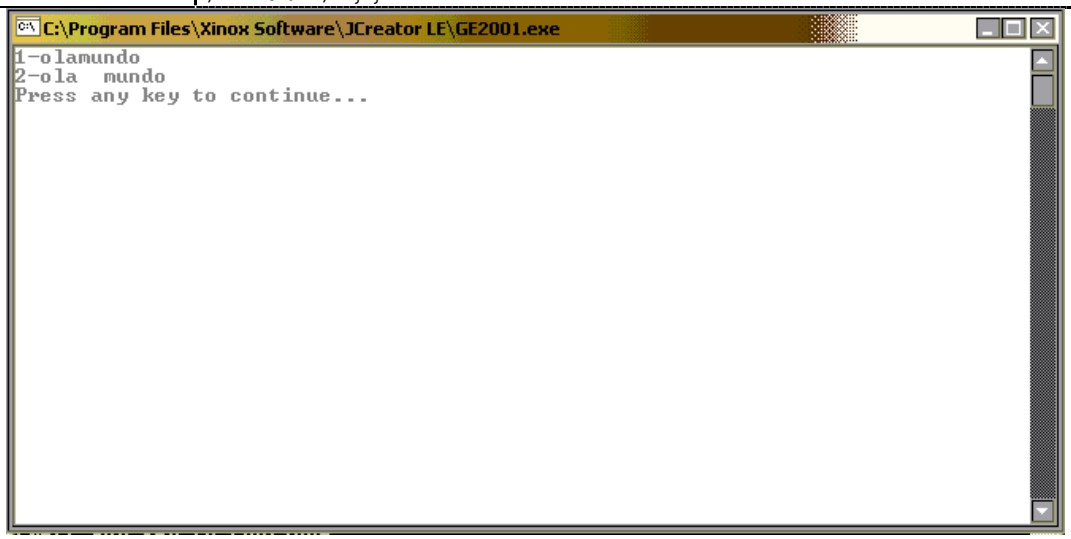


Figura 7 - Exemplo de Execução do programa concatenaString.java

10 - Como ler do Teclado?

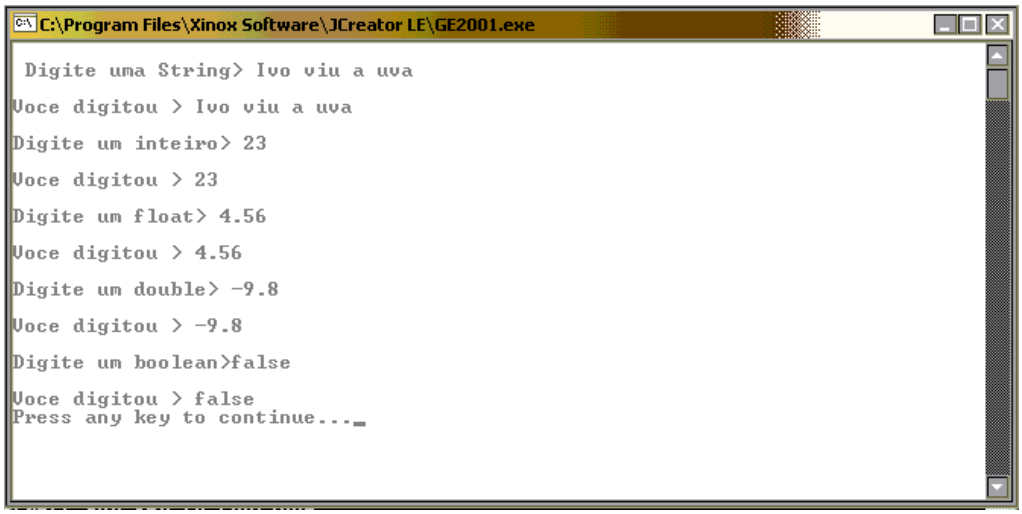
Entrada Texto

```
/*1.*/ // arquivo lerTeclado.java

/*2.*/ import java.io.*;
/*3.*/ public class lerTeclado
/*4.*/ {
/*5.*/   public String string()
/*6.*/   {
/*7.*/    byte[] b = new byte[128];
/*8.*/    try
/*9.*/    {
/*10.*/    System.in.read(b);
/*11.*/   }
/*12.*/   catch (java.io.IOException e)
/*13.*/           {System.out.println(e);}
/*14.*/   String saida = new String(b);
/*15.*/   return saida;
/*16.*/  }
/*17.*/  public double real()
/*18.*/  { return Double.parseDouble(string()); }

/*19.*/  public int inteiro()
/*20.*/  {
/*21.*/    return Integer.parseInt(string());
/*22.*/  }

/*23.*/  public static void main(String [] a)
/*24.*/  {
/*25.*/   lerTeclado entrada = new lerTeclado();

/*26.*/   System.out.println("Digite uma frase:");
/*27.*/   String s = entrada.string();
/*28.*/   System.out.println("Frase Lida=>"+s);

/*29.*/   System.out.println("Digite um numero:");
/*30.*/   double n = entrada.real();
/*31.*/   System.out.println("Numero Lido=>" + n);
/*32.*/  }
/*33.*/ }
```

Outra opção: a classe Scanner

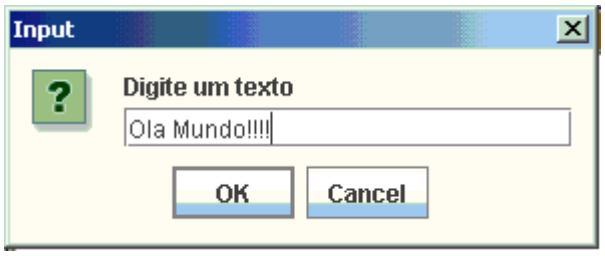
```
/*1.*/                // arquivo exemploScanner.java
/*2.*/ public class exemploScanner{
/*3.*/
/*4.*/ public static void main(String[] args)
/*5.*/ {
/*6.*/  java.util.Scanner in = new
/*7.*/       java.util.Scanner(System.in);


/*8.*/  System.out.print("\n Digite uma String> ");
/*9.*/  String v1 = in.nextLine();
/*10.*/ System.out.println("\nVoce digitou> " + v1);
```

```
/*11.*/  System.out.print("\nDigite um inteiro> ");
/*12.*/  int v2 = in.nextInt();
/*13.*/  System.out.println("\nVoce digitou> " + v2);

/*14.*/  System.out.print("\nDigite um float> ");
/*15.*/  float v3 = in.nextFloat();
/*16.*/  System.out.println("\nVoce digitou> " + v3);

/*17.*/  System.out.print("\nDigite um double> ");
/*18.*/  double v4 = in.nextDouble();
/*19.*/  System.out.println("\nVoce digitou> " + v4);

/*20.*/  System.out.print("\nDigite um boolean>");
/*21.*/  boolean v5 = in.nextBoolean();
/*22.*/  System.out.println("\nVoce digitou> " + v5);
/*23.*/ }
/*24.*/ }
```



Terceira opção: Janela de Leitura (JOptionPane)

```
/*1.*/  // arquivo testeTeclado1.java
/*2.*/  class testeTeclado1
/*3.*/  {
/*4.*/  public static void main(String [] a)
/*5.*/  { String m1;
/*6.*/  m1 = javax.swing.JOptionPane.showInputDialog
/*7.*/           ("Digite um texto");
/*8.*/    System.out.println(m1);
/*9.*/  } }
```



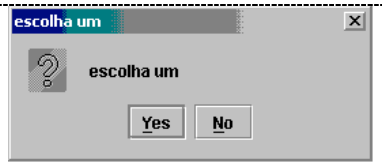
Outro exemplo leitura do teclado:

```
/*1.*/  // arquivo testeTeclado2.java
/*2.*/  class testeTeclado2
/*3.*/  {
/*4.*/  public static void main(String [] a)
/*5.*/  {
/*6.*/  int n1; double n2;
/*7.*/  n1 = Integer.parseInt(
/*8.*/  javax.swing.JOptionPane.showInputDialog(
/*9.*/  "Digite um numero"));
/*10.*/ n2 = Double.parseDouble(
/*11.*/ javax.swing.JOptionPane.showInputDialog(
/*12.*/ "Digite um outro numero"));
/*13.*/ javax.swing.JOptionPane.showMessageDialog
/*14.*/ ( null
/*15.*/  , n1 + "*" + n2 + "=" + n1*n2
/*16.*/  , "texto"
/*17.*/  ,
   javax.swing.JOptionPane.INFORMATION_MESSAGE);
/*18.*/   }}
```

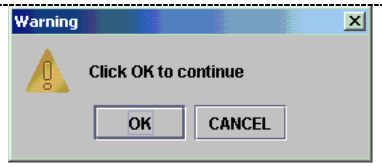
11 - A CLASSE JOptionPane

A classe javax.swing.JOptionPane permite apresentar uma caixa de mensagens ou solicitação. São exemplos do emprego da classe JOptionPane:

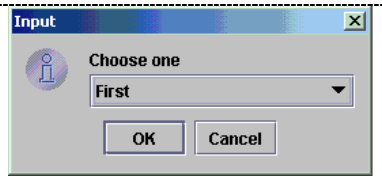
```
public static void exemplo1()
 {
  int escolha =
          JOptionPane.showConfirmDialog(null,
                 "escolha um", "escolha um",
                 JOptionPane.YES_NO_OPTION);
  if (escolha == 0) System.out.println(
        "Voce selecionou \"YES\" ");
  if (escolha == 1) System.out.println(
        "Voce selecionou \"NO\" ");
  if (escolha == -1) System.out.println(
        "Voce precisa selecionar uma opcao!");
 }
```




```
public static void exemplo2()
 {
  Object[] options = { "OK", "CANCEL" };

  int selecionado;
  selecionado = JOptionPane.showOptionDialog
       (null,"Click OK to continue"
       ,"Warning", JOptionPane.DEFAULT_OPTION,
       JOptionPane.WARNING_MESSAGE,
        null, options, options[0]);

  if (selecionado == 0) {System.out.println
                ("Voce selecionou OK");}
  else if (selecionado == 1)
          {System.out.println(
              "Voce selecionou CANCEL");}
       else {System.out.println(
           "Voce selecionou " + selecionado);}
 }
```




```
 public static void exemplo3()
 {
  Object[] possibleValues =
     { "First", "Second", "Third" };
  Object selectedValue =
         JOptionPane.showInputDialog(
                null,"Choose one", "Input"
```



```
            , JOptionPane.INFORMATION_MESSAGE
                , null, possibleValues
                , possibleValues[0]);
  if (selectedValue == possibleValues[0])
     {System.out.println(
               "Voce selecionou o primeiro");}
  if (selectedValue == possibleValues[1])
     {System.out.println(
               "Voce selecionou o segundo");}
  if (selectedValue == possibleValues[2])
     {System.out.println(
           "Voce selecionou o terceiro");}
 }
```

Exemplo do uso da classe JOptionPane.

```
/*1.*/   // arquivo testeJOptionPane.java

/*2.*/   import javax.swing.*;
/*3.*/   class testeJOptionPane
/*4.*/   {
/*5.*/   public static void exemplo1()
/*6.*/   {
/*7.*/   int escolha =
/*8.*/   JOptionPane.showConfirmDialog(null,
/*9.*/      "escolha um", "escolha um",
/*10.*/     JOptionPane.YES_NO_OPTION);
/*11.*/  if (escolha == 0) System.out.println(
/*12.*/       "Voce selecionou \"YES\" ");
/*13.*/  if (escolha == 1) System.out.println(
/*14.*/  "Voce selecionou \"NO\" ");
/*15.*/  if (escolha == -1) System.out.println(
/*16.*/     "Voce precisa selecionar uma opcao! ");
/*17.*/  }
/*18.*/  ////////////////////////////////////////
/*19.*/  public static void exemplo2()
/*20.*/  {
/*21.*/  Object[] options = { "OK", "CANCEL" };

/*22.*/  int selecionado;
/*23.*/  selecionado = JOptionPane.showOptionDialog
/*24.*/           (null,"Click OK to continue"
/*25.*/           ,"Warning", JOptionPane.DEFAULT_OPTION
/*26.*/           ,JOptionPane.WARNING_MESSAGE
/*27.*/           ,null, options, options[0]);

/*28.*/  if (selecionado == 0) {System.out.println
/*29.*/                    ("Voce selecionou OK");}
```

```
/*30.*/  else if (selecionado == 1) {System.out.println(
/*31.*/               "Voce selecionou CANCEL");}
/*32.*/        else {System.out.println(
/*33.*/              "Voce selecionou " + selecionado);}
/*34.*/  }
/*35.*/  ////////////////////////////////////////
/*36.*/  public static void exemplo3()
/*37.*/  {
/*38.*/  Object[] possibleValues =
/*39.*/              { "First", "Second", "Third" };
/*40.*/  Object selectedValue =
/*41.*/        JOptionPane.showInputDialog(
/*42.*/                          null,"Choose one", "Input"
/*43.*/                          ,JOptionPane.INFORMATION_MESSAGE
/*44.*/                          , null, possibleValues
/*45.*/                          , possibleValues[0]);
/*46.*/  if (selectedValue == possibleValues[0])
/*47.*/  {System.out.println(
/*48.*/             "Voce selecionou o primeiro");}
/*49.*/  if (selectedValue == possibleValues[1])
/*50.*/  {System.out.println(
/*51.*/              "Voce selecionou o segundo");}
/*52.*/  if (selectedValue == possibleValues[2])
/*53.*/               {System.out.println(
/*54.*/                 "Voce selecionou o terceiro");}
/*55.*/  }
/*56.*/  ////////////////////////////////////////////
/*57.*/  public static void main(String[] a)
/*58.*/  {
/*59.*/   exemplo1();
/*60.*/   exemplo2();
/*61.*/   exemplo3();
/*62.*/   JOptionPane.showMessageDialog(null
/*63.*/      ,"Fim do Programa");
/*64.*/  }
/*65.*/  }
```

Alguns Tipos de Janelas:

- **showConfirmDialog** : Pede uma confirmação com botões de yes/no/cancel.
- **ShowInputDialog** : Fornece um prompt para entrada do usuário.
- **ShowMessageDialog** :Informa ao usuário que alguma coisa aconteceu.
- **showOptionDialog** : A união das três acima.

Tipos de Mensagens:

- `ERROR_MESSAGE`
- `INFORMATION_MESSAGE`
- `WARNING_MESSAGE`
- `QUESTION_MESSAGE`
- `PLAIN_MESSAGE`

Tipos de Conjunto de Botões:

- `DEFAULT_OPTION`
- `YES_NO_OPTION`
- `YES_NO_CANCEL_OPTION`
- `OK_CANCEL_OPTION`

Exercícios:

1) Criar uma aplicação Java que calcule o fatorial de um numero. Esse número é lido através do teclado.
2) Criar uma aplicação Java que leia um número do teclado e verifique se esse número é par ou ímpar.

(dica: utilize o perador (%) que retorna o reto da divisao de dois numeros. Por exemplo:

```
if ( ( 50 % 5 ) == 0 ) // numero divisivel por 5.
```

12 - Reutilização de Classes.

Um dos aspectos positivos e de grande interesse na OO, é a possibilidade de reutilizarmos uma classe. Por exemplo, a classe Ponto2D, vista anteriormente, poderia ser utilizada para compor uma outra classe, a classe Reta a seguir.

```
/*1.*/   ////// ********************   arquivo Reta.java
/*2.*/   class Ponto2D {
/*3.*/   private double x,y;
/*4.*/   public void inicializaPonto2D(double _x,  double _y)
/*5.*/   { x = _x;
/*6.*/     y = _y;
/*7.*/   }

/*8.*/   public boolean ehIgual(Ponto2D outroPonto2D)
/*9.*/   {
/*10.*/  if ((x == outroPonto2D.x) &&
/*11.*/  (y == outroPonto2D.y))
/*12.*/  return true;
/*13.*/  else
/*14.*/  return false;
/*15.*/  }
/*16.*/  public Ponto2D origem()
/*17.*/  {
/*18.*/  Ponto2D temporario = new Ponto2D();
/*19.*/  temporario.inicializaPonto2D(0,0);
/*20.*/  return temporario;
/*21.*/  }
/*22.*/  public Ponto2D clona()
/*23.*/  {
/*24.*/  Ponto2D temporario = new Ponto2D();
/*25.*/  temporario.inicializaPonto2D(x,y);
/*26.*/  return temporario;
/*27.*/  }

/*28.*/  public String toString()
/*29.*/  {
/*30.*/   String resultado = "("+x+","+y+")";
/*31.*/   return resultado;
/*32.*/  }
/*33.*/  }
/*34.*/  /*************************************************************/
/*35.*/  public class Reta
```

```
/*36.*/  {
/*37.*/  Ponto2D p1 = new Ponto2D();
/*38.*/  Ponto2D p2 = new Ponto2D();

/*39.*/  public void setPontos(Ponto2D v1, Ponto2D v2)
/*40.*/  { setP1(v1);  setP2(v2); }

/*41.*/  public void setP1(Ponto2D v) {p1=v;}
/*42.*/  public void setP2(Ponto2D v) {p2=v;}

/*43.*/  public Ponto2D getP1() {return p1;}
/*44.*/  public Ponto2D getP2() {return p2;}

/*45.*/  public String toString()
/*46.*/  { String saida = "Reta formada pelos pontos " + p1.toString()
/*47.*/  + " e " + p2.toString();
/*48.*/  return saida;  }

/*49.*/  public static void main(String [] a)
/*50.*/  { Ponto2D pa = new Ponto2D();     Ponto2D pb = new Ponto2D();
/*51.*/  pa.inicializaPonto2D(3,4);  pb.inicializaPonto2D(7,8);
/*52.*/  Reta r = new Reta();
/*53.*/  r.setPontos(pa, pb);
/*54.*/  System.out.println(r);
/*55.*/  }}
```

13 - Construtores e Sobrecarga

O que são construtores?

Construtores são métodos especiais, que são chamados automaticamente quanto instâncias são criadas através da palavra-chave `new`.

Através desses construtores, pode-se garantir que o código que eles contém será executado antes de qualquer código em outros métodos, já que uma instância de uma classe só pode ser usada depois de ter sido criada com `new`.

Construtores são particularmente úteis para inicializar campos de instâncias de classes para garantir que, quando métodos dessas instâncias forem chamados, eles contenham valores específicos e não os *default*.

Como construtores são representados?

O construtor é um método que obrigatoriamente possui o mesmo nome da classe. Esse método não possui o valor de retorno informado. O exemplo abaixo apresenta, em negrito, um construtor para a classe Reta. Observe que o construtor modifica como a classe é instanciada (destaque em negrito e itálico no código a seguir).

Exemplo de definição de construtores:

```
/*1.*/   ////// ********************   arquivo Reta.java
/*2.*/   public class Reta
/*3.*/   {
/*4.*/   Ponto2D p1;
/*5.*/   Ponto2D p2;
/*6.*/   Public Reta(Ponto2D pA, Ponto2D pB)
/*7.*/   {p1=pA; p2=pB;}
/*8.*/   public void setPontos(Ponto2D v1, Ponto2D v2)
/*9.*/   {
/*10.*/  setP1(v1);
/*11.*/  setP2(v2);
/*12.*/  }
/*13.*/  public void setP1(Ponto2D v) {p1=v;}
/*14.*/  public void setP2(Ponto2D v) {p2=v;}
/*15.*/  public Ponto2D getP1() {return p1;}
/*16.*/  public Ponto2D getP2() {return p2;}
/*17.*/  public String toString()
/*18.*/  { String saida = "Reta formada pelos pontos "
/*19.*/     + p1.toString() + " e " + p2.toString();
/*20.*/     return saida; }
/*21.*/  public static void main(String [] a)
/*22.*/  {
/*23.*/  Ponto2D pa = new Ponto2D();
/*24.*/  Ponto2D pb = new Ponto2D();
/*25.*/  pa.inicializaPonto2D(3,4);
/*26.*/  pb.inicializaPonto2D(7,8);
/*27.*/  Reta r = new Reta(pa,pb);
/*28.*/  System.out.println(r); }}
```

Sobrecarga:

Em algumas situações, será útil ou interessante poder executar um método em uma classe passando mais ou menos argumentos.

Por exemplo, consideremos a classe Reta. Nessa classe, o construtor poderia ser chamado de duas formas. A primeira delas é fornecendo os dois pontos que definem a reta. Na Segunda, apenas um ponto e o outro ponto de definição poderia ser considerado a origem.

Apesar de terem o mesmo nome, essas duas assinaturas desse construtor são diferentes. Esse recurso é denominado de sobrecarga.

Observe, no código abaixo, a sobrecarga dos seguintes métodos: construtor, setPontos, setP1 e setP2.

Assinatura:

Java permite apenas uma única assinatura por método. Uma assinatura é formada por:

<nome do método> (<argumentos>).

Exemplo de sobrecarga de métodos.

```
/*1.*/   ////// ********************   arquivo Reta.java
/*2.*/   public class Reta
/*3.*/   {
/*4.*/   Ponto2D p1;
/*5.*/   Ponto2D p2;

/*6.*/   public Reta(Ponto2D pA, Ponto2D pB)
/*7.*/   {p1=pA; p2=pB;}

/*8.*/   public Reta(Ponto2D pA)
/*9.*/   {this(pA, pA.origem());}
/*10.*/  //observacao: "this(...)" chama outro construtor !!

/*11.*/  public void setPontos(Ponto2D v1, Ponto2D v2)
/*12.*/  {
/*13.*/   setP1(v1);
/*14.*/   setP2(v2);
/*15.*/  }

/*16.*/  public void setPontos(double x1, double y1
/*17.*/  , double x2, double y2)
/*18.*/  {
/*19.*/   setP1(x1,y1);
/*20.*/   setP2(x2,y2);
/*21.*/  }

/*22.*/  public void setP1(Ponto2D v) {p1=v;}
/*23.*/  public void setP2(Ponto2D v) {p2=v;}

/*24.*/  public void setP1(double x, double y)
/*25.*/  {p1= new Ponto2D(); p1.inicializaPonto2D(x,y);}

/*26.*/  public void setP2(double x, double y)
/*27.*/  {p2= new Ponto2D(); p2.inicializaPonto2D(x,y);}
```

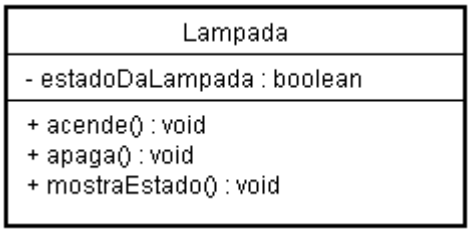
```
/*28.*/  public Ponto2D getP1() {return p1;}
/*29.*/  public Ponto2D getP2() {return p2;}

/*30.*/  public String toString()
/*31.*/  { String saida = "Reta formada pelos pontos "
/*32.*/       + p1.toString() + " e " + p2.toString();
/*33.*/  return saida; }

/*34.*/  public static void main(String [] a)
/*35.*/  { Ponto2D pa = new Ponto2D();
/*36.*/  Ponto2D pb = new Ponto2D();
/*37.*/  pa.inicializaPonto2D(3,4);
/*38.*/  pb.inicializaPonto2D(7,8);
/*39.*/  Reta r = new Reta(pa,pb);
/*40.*/  System.out.println(r);}
/*41.*/  }
```

A palavra reservada this(...) :

Um construtor não pode ser chamado diretamente. Java cria internamente para cada instância uma auto-referência, ou seja, uma referência à própria instância. Essa referência é representada pela palavra chave `this`.

Exercícios:

1) Escreva um construtor para a classe Lampada de forma que instâncias desta só possam ser criadas se um estado inicial for passado para o construtor. Esse estado pode ser o valor booleano que indica se a lâmpada está acesa (true) ou apagada (false).



2) Identifique e explique o(s) erro(s) na classe abaixo:

```
class Ponto2D
{
 private double x,y;
 Ponto2D()
 {
    Ponto2D(0.0,0.0);
 }
 Ponto2D(double coord1,double coord2)
 {
   x = coord1; y = coord2;
 }
} // fim da classe
```

3) Escreva uma classe denominada por Equation2g que permita calcular as raízes de uma equação do segundo grau.

$$a.x^2 + bx + c = 0$$

Escreva um construtor onde o usuário poderá fornecer os valores de a, b e c. e outro construtor onde o usuário não necessita fornecer valor algum.

4) Escreva dois construtores para a classe Ponto2D: um sem argumentos que considere que o ponto está na origem, ou seja, com coordenadas (0,0), e um que receba dois argumentos do tipo double e que os use para inicializar os campos da classe.

5) Escreva em Java a classe NumeroComplexo que represente um número complexo. A classe deverá ter os seguintes métodos:

- Construtor que obrigue o usuário fornecer a parte real e imaginária do número.
- **toString( )**
- **print( )**, que deve imprimir o número complexo encapsulado usando a notação *a*+*bi* onde *a* é a parte real e *b* a imaginária;
- **isEquals(NumeroComplexo)**, que recebe outra instância da classe NumeroComplexo e retorna true se os valores dos campos encapsulados forem iguais aos da instância passada como argumento;
- **somar(NumeroComplexo)**, que recebe outra instância da classe NumeroComplexo e soma este número complexo com o encapsulado usando a fórmula $(a+bi)+(c+di) = (a+c)+(b+d)i$;
- **subtrair(NumeroComplexo)**, que recebe outra instância da classe NumeroComplexo e subtrai o argumento do número complexo encapsulado usando a fórmula $(a+bi)-(c+di) = (a-c)+(b-d)i$;
- **multiplicar(NumeroComplexo)**, que recebe outra instância da classe NumeroComplexo e multiplica este número complexo com o encapsulado usando a fórmula $(a+bi) * (c+di) = (ac-bd)+(ad+bc)i$;
- **dividir(NumeroComplexo)**, que recebe outra instância da classe NumeroComplexo e divide o número encapsulado pelo passado como argumento usando a fórmula $\frac{(a+bi)}{(c+di)} = \frac{(ac+bd)}{c^2+d^2} + \frac{(bc-ad)}{c^2+d^2}i$

6) Escreva três construtores para a classe NumeroComplexo. Um construtor deverá receber os dois valores (real e imaginário) como argumentos, o outro somente o valor real, considerando o imaginário como sendo zero, e o terceiro construtor não recebe argumentos, considerando as partes real e imaginária do número complexo como sendo iguais a zero.

7) Modifique a classe Equation2g para que esta possa calcular o valor de raízes complexas como o exemplo abaixo:

$$1.x^2 + 1.x + 2 = 0$$

A solução é calculada da seguinte forma:

$$x_{1,2} = \frac{-1 \pm \sqrt{1^2 - 4*1*2}}{2*1} = \frac{-1 \pm \sqrt{-7}}{2} = \frac{-1 \pm 2.6457\sqrt{-1}}{2}$$

$$x_{1,2} = \frac{-1}{2} \pm i\frac{2.6457}{2}$$

14 - Campos e Métodos Estáticos.

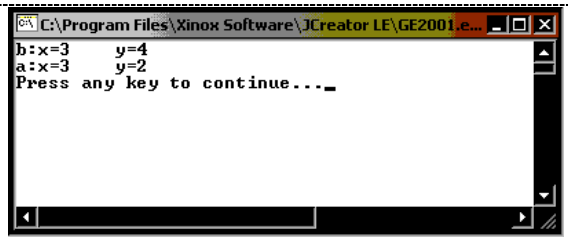
Introdução:

Campos estáticos em uma classe são compartilhados por todas as instâncias dessa classe. Em outras palavras, somente um valor será armazenado em um campo estático, e caso este valor seja modificado por uma das instâncias dessa classe, a modificação será refletida em todas as outras instâncias dessa classe.

Exemplo de Campo Estático:

```
/*1.*/  //*************** arquivo testeEstatico.java
/*2.*/  class Estatico
/*3.*/  {
/*4.*/  private static int x;
/*5.*/   private int y;
/*6.*/   public void setXY(int novoX, int novoY)
/*7.*/   { setX(novoX); setY(novoY);}
/*8.*/   public void setX(int novoX) {x=novoX;}
/*9.*/   public void setY(int novoY) {y=novoY;}
/*10.*/  public String toString()
/*11.*/  {return ("x="+ x + "\t y="+y); }
/*12.*/  public void print()
/*13.*/  { System.out.println(toString()); }
/*14.*/ }
/*15.*/ public class testeEstatico
/*16.*/ { public static void main(String[] args)
/*17.*/   {  Estatico a = new Estatico();
/*18.*/      Estatico b = new Estatico();
/*19.*/      a.setXY(1,2);
/*20.*/      b.setXY(3,4);
/*21.*/      System.out.println("b:"+ b);
/*22.*/      System.out.println("a:"+ a);
/*23.*/  }
/*24.*/   }
```

Resultado da Execução:



Métodos Estáticos:

Métodos estáticos em uma classe são declarados com o modificador `static`, que deve preceder o tipo de retorno do método.

A diferença principal entre métodos estáticos e métodos não-estáticos é que os métodos estáticos podem ser chamados sem a necessidade de criação de instâncias das classes às quais pertencem.

Métodos estáticos são adequados para:

- implementar bibliotecas de rotinas que sejam independentes de dados armazenados em classes, ou seja, métodos que só necessitem dos dados passados como argumentos para efetuar a tarefa requerida.
- Se o método public static void main(String[ ] args) desejar fazer, de maneira direta, uma chamada a uma função da classe, essa função deevrá ser definida como estática.

Exemplo de definição de biblioteca:

```
                                        /*  arquivo BatCinto.java*/
/*1.*/  import javax.swing.*;
/*2.*/  /*
/*3.*/  Na pratica de programacao, em algumas situacoes,
/*4.*/     repetimos processos.
/*5.*/  Nesta classe, defino algumas funcoes que poderiam
/*6.*/    ser empregadas para facilitar a pratica
/*7.*/    de programacao.
/*8.*/  */
/*9.*/  public class BatCinto
/*10.*/ {
/*11.*/  public static String leiaTexto(String texto)
/*12.*/  { java.util.Scanner teclado;
/*13.*/    teclado = new java.util.Scanner(System.in);
/*14.*/    System.out.println(texto);
/*15.*/    String saida = teclado.nextLine();
/*16.*/    return saida;
/*17.*/   }

/*18.*/ public static double leiaDouble(String texto)
/*19.*/ {  String lido = leiaTexto(texto);
/*20.*/    double saida=-666;
/*21.*/    try {saida  = Double.parseDouble(lido);}
/*22.*/    catch (java.lang.NumberFormatException e)
/*23.*/    {System.out.println("\n");
/*24.*/     for (int i = 0; i < 50; i++)
/*25.*/            System.out.print("*");
/*26.*/     System.err.println("\n Numero Invalido!");
/*27.*/     System.err.println("\n Erro de conversao:"
```

```
                                + e);
/*28.*/  /* System.err => para impressao de mensagem de
   erro*/
/*29.*/      System.err.println("\n");
/*30.*/    }
/*31.*/    return saida;
/*32.*/ }

/*33.*/ public static int leiaInteiro(String texto)
/*34.*/ {  String lido = leiaTexto(texto);
/*35.*/    int saida=-666;
/*36.*/    try
/*37.*/      { saida = (new Integer(lido)).intValue();
/*38.*/ /* outra maneira de fazer o
   Integer.parseInt(String)*/
/*39.*/      }
/*40.*/    catch (java.lang.NumberFormatException e)
/*41.*/    { System.out.println("\n");
/*42.*/      for (int i = 0; i < 50; i++)
/*43.*/            System.out.print("*");
/*44.*/      System.err.println("\n Numero Invalido!");
/*45.*/      System.err.println("\n Erro de conversao:"
                          + e);
/*46.*/ /* System.err => para impressao de mensagem de
   erro*/
/*47.*/      System.err.println("\n");
/*48.*/ }
/*49.*/     return saida;
/*50.*/ }

/*51.*/ public static String formataNumero(double numero
                                       , int precisao)
/*52.*/ {  /* numero=> é o numero a ser formatado.
/*53.*/            Por exemplo 3.4678912
/*54.*/       precisao=> informa quantas casas depois da
/*55.*/             virgula devem ser apresentadas. */
/*56.*/   Java.text.NumberFormat saida ;
/*57.*/   saida = java.text.NumberFormat.getInstance();
/*58.*/   saida.setMaximumFractionDigits(precisao);
```

```
/*59.*/   return saida.format(numero);
/*60.*/ }
/*61.*/ /*////////////////////////////////////
/*62.*/ ////////////////////////////////////
/*63.*/ ### janelas que poderiam ser uteis: ##
/*64.*/ ////////////////////////////////////
/*65.*/ ////////////////////////////////////*/

/*66.*/ public static void janelaMostraTexto(String t)
/*67.*/ { JOptionPane.showMessageDialog(null,t);}

/*68.*/ public static String janelaLeiaTexto(
/*69.*/            String texto)
/*70.*/ {  String saida;
/*71.*/    saida = JOptionPane.showInputDialog(texto);
/*72.*/    return saida;
/*73.*/ }

/*74.*/ public static int janelaLeiaInteiro(String texto)
/*75.*/ {  String aux = janelaLeiaTexto(texto);
/*76.*/    int saida=-666;
/*77.*/    try{ saida = Integer.parseInt(aux);}
/*78.*/    catch (java.lang.NumberFormatException e)
/*79.*/     { System.out.println("\n");
/*80.*/       for (int i = 0; i < 50; i++)
/*81.*/            System.out.print("*");
/*82.*/       System.err.println("\n Numero Invalido!");
/*83.*/       System.err.println("\n Erro de conversao:"
                          + e);
/*84.*/ /* System.err => para impressao de mensagem de
   erro*/
/*85.*/       System.err.println("\n");
/*86.*/      }
/*87.*/    return saida;
/*88.*/ }

/*89.*/ public static double janelaLeiaDouble(
/*90.*/                                  String texto)
/*91.*/ {  String aux = janelaLeiaTexto(texto);
```

```
/*92.*/    double saida=-666;
/*93.*/    try {saida = Double.parseDouble(aux);}
/*94.*/    catch (java.lang.NumberFormatException e)
/*95.*/    { System.out.println("\n");
/*96.*/      for (int i = 0; i < 50; i++)
/*97.*/          System.out.print("*");
/*98.*/      System.err.println("\n Numero Invalido!");
/*99.*/      System.err.println("\n Erro de conversao:" +
   e);
/*100.*//* System.err => para impressao de mensagem de
                        erro*/
/*101.*/     System.err.println("\n");
/*102.*/   }
/*103.*/   return saida;
/*104.*/}

/*105.*/public static String janelaCaixaDeSelecao(
                  String texto, String[] opcoes)
/*106.*/{
/*107.*/  Object[] valoresPossiveis;
/*108.*/  valoresPossiveis = new Object[opcoes.length];
/*109.*/  for (int i = 0; i < opcoes.length; i++)
/*110.*/    valoresPossiveis[i] = opcoes[i];

/*111.*/  Object selecionado =
/*112.*/         JOptionPane.showInputDialog(null
/*113.*/                ,texto, "Escolha uma opção"
/*114.*/                , JOptionPane.INFORMATION_MESSAGE
/*115.*/                , null, valoresPossiveis
/*116.*/                , valoresPossiveis[0]);
/*117.*/  String saida = null;
/*118.*/  for (int i = 0 ; i < opcoes.length; i++)
/*119.*/  if (selecionado == valoresPossiveis[i])
/*120.*/      {saida = opcoes[i]; break;}
/*121.*/  return saida;
/*122.*/ }
/*123.*/}

/*////////////////////////////////////////////
```

```
            Exemplo do uso da biblioteca:
////////////////////////////////////////////////*/

/*124.*/class testeBatCinto
/*125.*/{
/*126.*/ public static void main(String[] a)
/*127.*/ {
/*128.*/ int x = BatCinto.janelaLeiaInteiro(
            "Digite um numero inteiro");

/*129.*/ double z = BatCinto.janelaLeiaDouble(
                 "Digite um numero real");

/*130.*/ BatCinto.janelaMostraTexto("Li:" + x
                       + "\nLi tambem:" + z);

/*131.*/  int u = BatCinto.leiaInteiro(
                   "Digite um numero inteiro");
/*132.*/ double k = BatCinto.leiaDouble(
                      "Digite um numero real");

/*133.*/ String[] frutas = {"Acerola", "Limao", "Laranja"
                           , "Pessego", "uva"};
/*134.*/ String escolheFruta =
            BatCinto.janelaCaixaDeSelecao(
              "Qual eh a fruta que voce mais gosta?"
              , frutas);

/*135.*/ BatCinto.janelaMostraTexto(
                     "Que coincidencia! "
                     + escolheFruta
                     + " tambem eh a minha predileta.");
/*136.*/ System.out.println("\n 1.345678934 “
             + “ com tres casas de precisao"
             + BatCinto.formataNumero(1.345678934,3));
/*137.*/ }
/*138.*/}
```

Métodos chamados diretamente do método main:

Se um método for chamado diretamente a partir do método **main**, este método deverá ser **obrigatoriamente** declarado como estático. O exemplo testeResistor a seguir ilustra essa necessidade.

Exemplo de Métodos chamados diretamente do método main:

```
/*1.*/   //******************* arquivo testeResistor2.java
/*2.*/  // definicao da Aplicacao
/*3.*/  class testeResistor2
/*4.*/  {  static double calculaSerie( double r1
/*5.*/                                 ,double r2)
/*6.*/     {return r1+r2;}

/*7.*/     static double calculaParalelo(double r1
/*8.*/                                 ,double r2)
/*9.*/     { double saida=-1.0;
/*10.*/     if ((r1+r2) != 0.0)
/*11.*/        saida = r1*r2/(r1+r2);
/*12.*/     return saida;
/*13.*/    }

/*14.*/  public static void main(String [] arg)
/*15.*/  { double calculo =
/*16.*/      calculaParalelo(1.0,1.0);
/*17.*/   double calculo2 =
/*18.*/      calculaSerie(1.0,1.0);
/*19.*/   System.out.println( calculo
/*20.*/            + "\n" + calculo2);
/*21.*/  }
/*22.*/ }
```

Outro exemplo:

```
/*1.*/  // arquivo CalculoDePrecoDeTerreno.java
/*2.*/  class CalculoDePrecoDeTerreno  {
/*3.*/  public static void main(String[] argumentos)
/*4.*/  {
/*5.*/   double preco;
/*6.*/   System.out.print(
/*7.*/        "O preco do terreno N1 eh ");
/*8.*/   preco = precoDoTerreno(450,1);
/*9.*/   System.out.println(preco);
/*10.*/  System.out.print(
/*11.*/       "O preco do terreno Q2 eh ");
```

```
/*12.*/  preco = precoDoTerreno(475,4);
/*13.*/  System.out.println(preco);
/*14.*/  System.out.print(
/*15.*/      "O preco do terreno F3 eh ");
/*16.*/  System.out.println(precoDoTerreno(525,2));
/*17.*/ }

/*18.*/ public static double precoDoTerreno
/*19.*/         (double area,int localizacao)
/*20.*/ { double preco = 0;
/*21.*/   if (localizacao == 1) preco = area*22.00;
/*22.*/   if (localizacao == 2) preco = area*27.00;
/*23.*/   if (localizacao == 3) preco = area*29.50;
/*24.*/   if (localizacao == 4) preco = area*31.50;
/*25.*/   if (localizacao == 5) preco = area*34.50;
/*26.*/   return preco;
/*27.*/ }
/*28.*/    }
```

Exercícios:

1) Escreva a classe ConversaoDeUnidadesDeArea com métodos estáticos para conversão das unidades de área segundo a lista abaixo.

- 1 metro quadrado = 10.76 pés quadrados
- 1 pé quadrado = 929 milhas quadradas
- 1 milha quadrada = 640 acres
- 1 acre = 43.560 pés quadrados

2) Escreva a classe ConversaoDeUnidadesDeTempo com métodos estáticos para conversão aproximada das unidades de segundo a lista abaixo.

- 1 minuto = 60 segundos
- 1 hora = 60 minutos
- 1 dia = 24 horas
- 1 semana = 7 dias
- 1 mês = 30 dias
- 1 ano = 365.25 dias

15 Leitura e Escrita em Arquivo Texto

Introdução:

Java não possui uma função específica para entrada e saída de textos em arquivo. O exemplo a seguir apresenta como essa entrada e saída poderia ser implementada em Java.

Nesse exemplo, uma classe chamada Arquivo é criada. O construtor da classe obriga o seu usuário fornecer o nome do aquivo ou instância da classe File. Duas funções úteis estão a disposição: `ler()` e `escrever(String)`.

Exemplo de Entrada e Saída de Textos:

```
/*1.*/  ////////////////////  arquivo exemploIO.java
/*2.*/  import java.io.*;

/*3.*/  class Arquivo
/*4.*/  { private File name;

/*5.*/    public Arquivo(String nome)
/*6.*/    { name= new File(nome); }

/*7.*/    public Arquivo(File v){name=v;}

/*8.*/    public String ler()
/*9.*/    { String arq = "Arquivo lido:"
/*10.*/       +name.getName() +"\n";
/*11.*/     if (name.exists())
/*12.*/     {
/*13.*/       if (name.isFile())
/*14.*/       {
/*15.*/        try
/*16.*/        {
/*17.*/        RandomAccessFile r = new
/*18.*/        RandomAccessFile(name,"r");
/*19.*/        StringBuffer buf = new StringBuffer();
/*20.*/        String text;
/*21.*/        while ( (text=r.readLine()) != null)
/*22.*/               arq += text + "\n";
/*23.*/         }
/*24.*/        catch (IOException erro)
/*25.*/           {System.out.println(
/*26.*/          "***ERRO: Nao foi possivel ler \n\t"
/*27.*/               + erro);}
/*28.*/          }
/*29.*/     }
/*30.*/     return arq;
/*31.*/   }

/*32.*/   public void escrever(String texto)
/*33.*/   {
/*34.*/     try{
/*35.*/      if (name.exists()) {name.delete();}
/*36.*/      RandomAccessFile r = new
/*37.*/      RandomAccessFile(name,"rw");
/*38.*/      StringBuffer buf = new StringBuffer();
/*39.*/      r.writeBytes(texto);
/*40.*/      }
/*41.*/      catch (IOException erro)
/*42.*/      {System.out.println(
/*43.*/         "***Erro de Escrita:\n\t" + erro); }
/*44.*/    }
```

```
/*45.*/
/*46.*/   public static File escolha()
/*47.*/   {
/*48.*/    javax.swing.JfileChooser escolhaArq =
/*49.*/       new javax.swing.JFileChooser(".");
/*50.*/    escolhaArq.showOpenDialog(null);
/*51.*/    return escolhaArq.getSelectedFile();
/*52.*/   }
/*53.*/  }

/*54.*/public class exemploIO
/*55.*/{
/*56.*/ public static void main(String[] arg)
/*57.*/ {
/*58.*/ Arquivo arq= new Arquivo("c://exemplo.txt");

/*59.*/ arq.escrever("Ola Mundo!\n Numero Gerado:"
/*60.*/   + Math.random());

/*61.*/ String textoLido = arq.ler();
/*62.*/ System.out.println(textoLido);
/*63.*/ Arquivo arq2 =
/*64.*/         new Arquivo(Arquivo.escolha());
/*65.*/ String conteudo = arq2.ler();
/*66.*/ System.out.println(conteudo);
/*67.*/ }
/*68.*/   }
```

16 Herança

Introdução:

Durante o processo de representação de objetos ou entidades no ambiente computacional, pode-se observar algumas características comuns entre esses objetos.

Por exemplo, em um ambiente acadêmico, duas entidades podem ser facilmente identificadas, **Aluno** e **Professor**.

Tanto Aluno, como Professor possuem atributos comuns a qualquer pessoa tais como: nome, data de nascimento, telefone e e-mail.

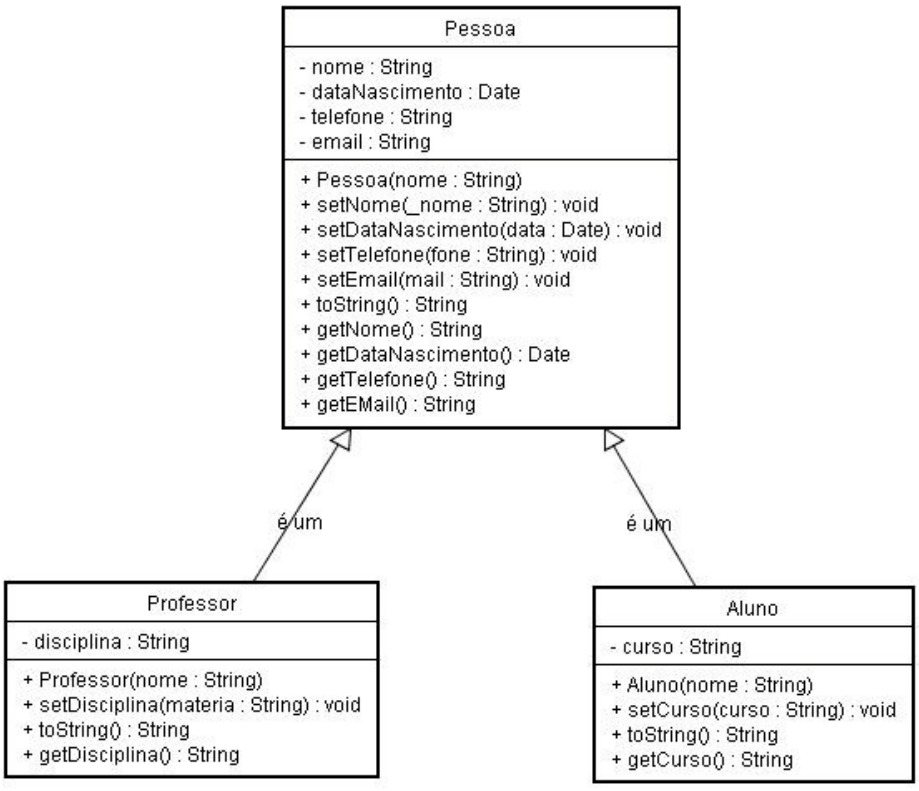
Poderíamos dizer que esses atributos comuns são associados a uma outra entidade chamada **Pessoa**. Dessa forma, pode-se afirmar que **Aluno** e **Professor** compartilham essa estrutura de informações. Ou que **Aluno** é uma especialização de **Pessoa** e **Professor** também é uma especialização de **Pessoa**.

Aluno e Professor definem atributos específicos de cada entidade. Um atributo específico do aluno é o curso que ele está matriculado, por exemplo. Um atributo específico do professor é a disciplina que ele ministra, por exemplo.

Representa-se Esse tipo de estrutura é representada por meio de ***Herança***. Uma classe é dita "classe filha", quando essa herda a estrutura de uma outra classe, denominada de "classe pai". Por exemplo, pode-se dizer que Pessoa é a "classe pai" de "Aluno" e "Professor".

A figura a seguir ilustra essa representação em linguagem UML.

Exemplo de representação da estrutura de Herança:



Essa estrutura de herança pode ser denominada de estrutura "é um". Ou seja, Professor "**é um**"(a) Pessoa, Aluno "**é um**"(a) Pessoa.

Representando os dados nesse tipo de estrutura, há a possibilidade de compartilhamento de dados e métodos entre as classes "pai" e "filha".

As classes "pai" e "filha" também são denominadas de "superclasse" e

"classe".

Código parcial da classe Pessoa:

```
// arquivo Pessoa.java
import java.util.Date;

public class Pessoa {
   private String nome;
   private Date dataNascimento;
   private String telefone;
   private String email;

   public Pessoa(String nome) {  }

   public void setNome(String _nome) {     }

   public void setDataNascimento(Date data) {}

   public void setTelefone(String fone) {  }

   public void setEmail(String mail) {     }

   public String toString() {return null;  }

   public String getNome() {return null;   }

   public Date getDataNascimento(){return null;}

   public String getTelefone() { return null;}

   public String getEMail() {return null;  }
}
```

Código parcial da classe filha Aluno:

```
// arquivo Aluno.java
public class Aluno extends Pessoa {
   private String curso;
   public Aluno(String v)
             { super(v);     }
   public void setCurso(String _curso) {}
   public String toString() {return null;  }
   public String getCurso() {return null;  }
}
```

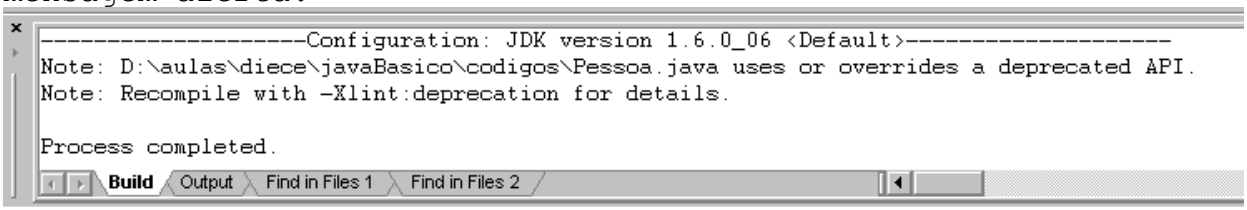
Código parcial da classe filha Professor:

```
// arquivo Professor.java
public class Professor extends Pessoa {
   private String disciplina;
   public Professor(String v)
    {super(v);     }
   public void setDisciplina(String _materia){}
   public String toString() {return null;}
   public String getDisciplina() {return null;}
}
```

Código completo da classe Pessoa:

```
/*1.*/  // arquivo Pessoa.java

/*2.*/  import java.util.*;

/*3.*/  public class Pessoa {
/*4.*/  private String nome;
/*5.*/  private Date dataNascimento;
/*6.*/  private String telefone;
/*7.*/  private String email;

/*8.*/  public Pessoa(String v) { setNome(v);}

/*9.*/  public void setNome(String v) { nome = v; }

/*10.*/ public void setDataNascimento(int dia, int mes
                                ,int ano)
/*11.*/ { dataNascimento =
/*12.*/ new Date((ano-1900),(mes-1),dia); }

/*13.*/ public void setTelefone(String fone)
/*14.*/ { telefone = fone;}

/*15.*/ public void setEmail(String v)
/*16.*/ { email = v; }

/*17.*/ public String toString()
/*18.*/ {  String saida="\nDados Pessoais:";
/*19.*/    saida += "\n\tNome:"+ getNome();
/*20.*/    saida += "\n\tNascimento:"+ getDataNascimento();
/*21.*/    saida += "\n\ttelefone:"+ getTelefone();
/*22.*/    saida += "\n\tE-Mail:"+ getEMail();
/*23.*/    return saida;  }

/*24.*/ public String getNome() {return nome;   }
/*25.*/ public Date getDataNascimento()
/*26.*/  {return dataNascimento;     }
/*27.*/ public String getTelefone() {return telefone;}
/*28.*/ public String getEMail() {   return email;}
/*29.*/ }
```

/* O código anterior, se compilado, fornecerá a seguinte mensagem alerta:



A classe Date utilizada no programa é uma classe antiga e recomenda-se utilizar a classe Calendar. A classe Calendar não foi utilizada por ser mais complexa que a classe Date e não é relevante para o exercício.
*/

Código Completo da classe Professor:

```
// arquivo Professor.java
public class Professor extends Pessoa {
  private String disciplina;

  public Professor(String v) {super(v);}

  public void setDisciplina(String v)
                        {disciplina=v;   }

  public String toString() {
      String saida=super.toString();
      saida += "\nDados Acadêmicos do Professor:";
      saida += "\n\tDisciplina: ";
      saida += getDisciplina();
      return saida;
     }

     public String getDisciplina()
                  {return disciplina;}
}
```

A palavra `super`:

Sempre que uma classe filha precisar se referenciar a uma classe pai, a palavra `super` pode ser usada. Na classe Professor, essa palavra é utilizada no construtor para chamar o contrutor da classe pai. Nesse caso, **" `super( parâmetros );` ".**

Uma classe filha chama um método da classe pai através da seguinte sintaxe de instrução:

`super.<nome do método da classe pai>`

Por exemplo, o método `toString()` da classe Professor chama o método `toString()` da classe Pessoa através da instrução `super.toString()`.

Código do programa de teste do exemplo considerado.

```
// arquivo testeAcademico.java
import javax.swing.*;
import java.util.*;


public class testeAcademico
{
 public static void main(String args[])
 {   Professor p1 = novoProfessor();
     System.out.println(p1); }
// método de auxílio
 public static Professor novoProfessor(){
 String nome =  JOptionPane.showInputDialog
                 ("Digite o nome");
 int ano = Integer.parseInt(
                JOptionPane.showInputDialog
               ("Digite o ano de nascimento"));
 int mes =  Integer.parseInt(
                JOptionPane.showInputDialog
```

```
                ("Digite o mes de nascimento"));
int dia = Integer.parseInt(
                JOptionPane.showInputDialog
                ("Digite o dia de nascimento"));

String fone = JOptionPane.showInputDialog
                ("Digite o numero de telefone");
String mail = JOptionPane.showInputDialog
                ("Digite o email");
String disciplina = JOptionPane.showInputDialog
           ("Digite a Disciplina do Professor");

 Professor saida = new Professor(nome);
 saida.setDataNascimento(dia,mes,ano);
 saida.setTelefone(fone);
 saida.setEmail(mail);
 saida.setDisciplina(disciplina);
 return saida;
} }
```

Comentários: O método setDataNascimento utilizado no programa anterior, escrito em caracter itálico e sublinhado, faz referência ao método da classe pai e não o método da classe Professor.

Como evitar que um determinado método seja sobre-escrito?.

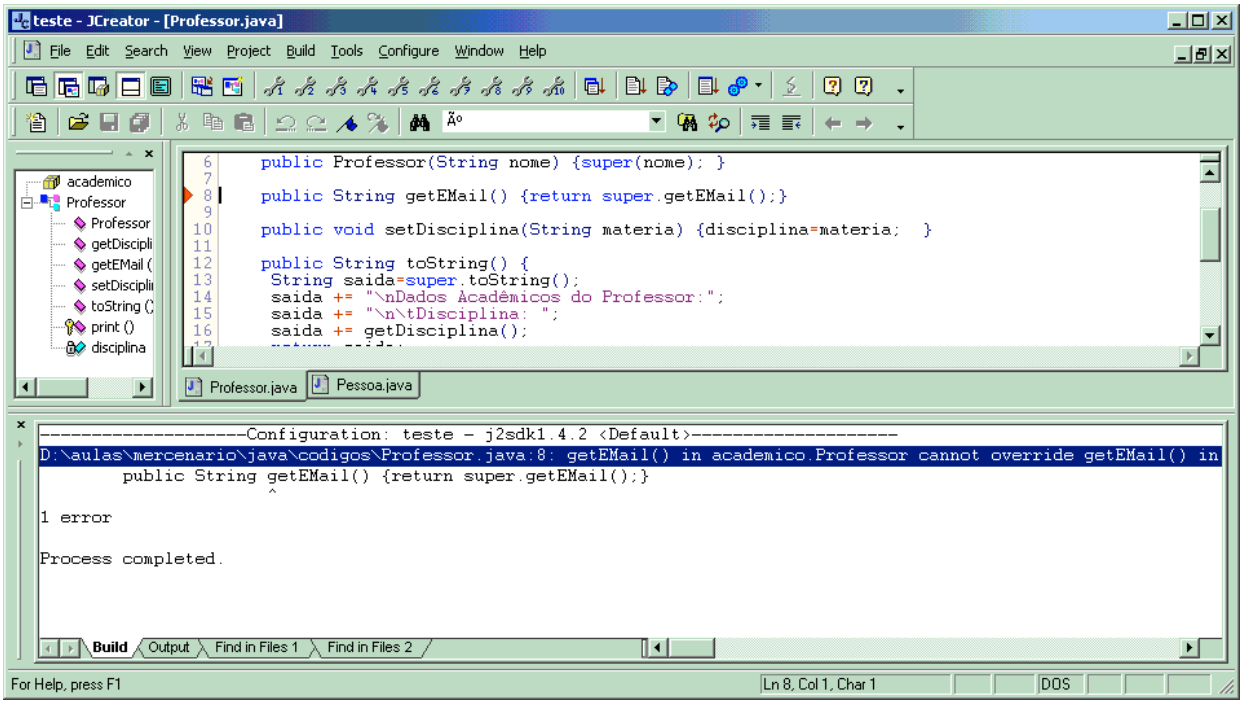
O modificador final inserido na assinatura do método define que o método não poderá ser sobre-escrito por nenhuma classe filha.

O modificador final, ilustrado abaixo, no método getEmail() da classe Pessoa define que esse método não pode ser sobre-escrito nas classes filhas.

```
41      public final String getEMail() {
42          return email;
43      }
44  }
```

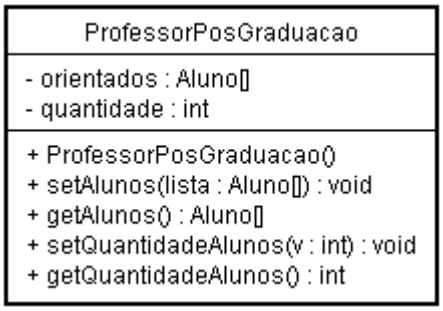
Pessoa.java *

Qualquer tentativa de (re)definição desse método em uma classe filha será considerada como erro. Por exemplo, a classe Professor, se compilada fornecerá a seguinte mensagem de erro:



Exercício:

1) Implementar a classe Aluno.
2) Implementar a classe Ponto3D que deverá ser "filha" da classe Ponto2D. A classe Ponto3D deverá armazenar três coordenadas.
3) Criar uma classe ProfessorPosGraduacao. Essa classe deverá ser filha da classe Professor e possuir a estrutura definida no diagrama UML ao lado.



17 - Interfaces

Introdução:

Um dos aspectos interessantes da OO é a capacidade de herança múltipla, ou seja, uma classe ser classe filha de duas outras classes.

Infelizmente Java não permite essa representação. Ou seja, o seguinte exemplo de código NÃO É ACEITO pelo compilador java:

public class MinhaClasse extends classePai1, classePai2, classePai3

Como alternativa a essa limitação, o Java permite sim, a implementação de classes "promessa". Uma classe "promessa" é denominada em Java como *interface*.

O que é uma *interface*?

É uma promessa de que uma classe herdeira implementará certos métodos com certas características.

Uma interface define apenas as assinaturas dos métodos e não os códigos. Esses códigos serão definidos pelas classes que a implementarem.

Uma interface não possui atributos. Todo e qualquer atributo definido em uma interface é tratado como constante.

Exemplo de Herança Múltipla

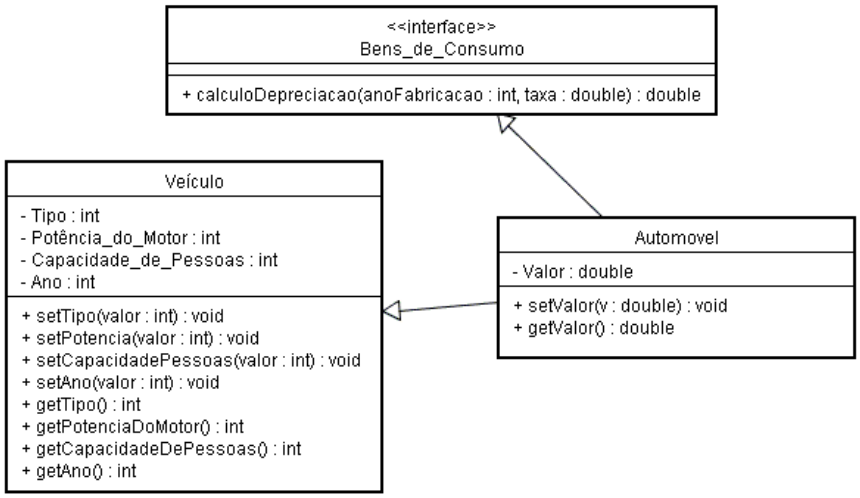
Como representar uma classe Automovel em Java?

Uma classe poderia representar a entidade veículo. Nesse contexto, por exemplo, um veículo pode ser do tipo Terrestre, Marítimo ou Aéreo ; pode ser movido por um motor que possui uma potência definida e possui um ano de fabricação.

Outra classe poderia representar a entidade "Bens de Consumo", por exemplo. Nessa classe, a informação importante destacada aqui é o Cálculo de Depreciação do bem de consumo.

Pode-se então representar uma classe Automovel como sendo filha das classes Veículo e Bens_de_Consumo, conforme ilustra o diagrama abaixo.

Exemplo de Representação de uma interface



Observe que nesse diagrama, a interface possui a referência <<interface>> antes do seu nome.

Exemplo de interface:

```
//////////////   arquivo Bens_de_Consumo.java
public interface Bens_de_Consumo
{
// metodo sem o corpo. Apenas a assinatura!
  public double calculoDepreciacao(
                  int anoFabricacao
                , double taxa);
}
//// fim do arquivo !!
```

Exemplo de classe:

```
//////////////////   arquivo Veiculo.java
public class Veiculo {
      /**
       *  Qual o seu tipo?
       *  0 -> Maritmo,
       *  1 -> Terrestre,
       *  2 -> Aereo
       */
      private int Tipo=0;
      private int Potencia_do_Motor;
      private int Capacidade_de_Pessoas;
      private int Ano;

      public void setTipo(int valor)
        {if (  (valor ==0)
             ||(valor ==1)
             ||(valor ==2)) Tipo = valor;
        }
      public void setPotencia(int valor)
        {Potencia_do_Motor = valor;}
      public void setCapacidadePessoas(int valor)
        { Capacidade_de_pessoas = valor;}
      public void setAno(int valor)
        {Ano = valor;}

      public int getTipo() {return Tipo;}
      public int getPotenciaDoMotor()
         { return Potencia_do_Motor;   }
      public int getCapacidadeDePessoas()
         { return Capacidade_de_Pessoas;}
      public int getAno() { return Ano;}
}
```

Exemplo da classe que implementa uma interface:

```
//////////////////////   arquivo Automovel.java
public class Automovel extends Veiculo
                       implements Bens_de_Consumo
{
      private double Valor;

      public double calculoDepreciacao(
                         int anoFabricacao
                       , double taxa)
      {
       java.util.Date hoje = new java.util.Date();
       int anoHoje = 1900+hoje.getYear();
        return (Valor*(1-(Math.abs(anoHoje-anoFabricacao)
                *(1-taxa/100))/100));
      }

      public void setValor(double v) {Valor=v;}

      public double getValor() { return Valor;}
}
```

Comentários:

O método **calculoDepreciacao(int anoFabricacao, double taxa)** definido na interface Bens_De_Consumo necessita obrigatoriamente ser definido na classe que a implementa, nesse caso, a classe Automóvel.

Uma classe pode implementar múltiplas interfaces. De maneira genérica, exemplifica-se:

public class MinhaClasse implements interface1, interface2, interface3

Comentários da Compilação

Durante a compilação, a mensagem de advertência da figura a seguir aparecerá. Essa mensagem refere-se à classe Date que é, atualmente, uma classe *deprecated*, ou seja, existe outra classe melhor para representar datas. Apesar disso, a classe Date pode ser utilizada sem maiores problemas.

```
--------------------Configuration: j2sdk1.4.2 <Default>--------------------
D:\aulas\diece\java\interfaces\Automovel.java:6: warning: getYear() in java.util.Date has been deprecated
                int anoHoje = hoje.getYear();
                              ^
1 warning
```

Métodos com a mesma assinatura

O que acontece se, por exemplo, a classe Veiculo possuir um método com a mesma assinatura de um método da interface Bens_de_Consumo?

Para responder a essa questão,

1- Insira na classe Veiculo o seguinte método:

public void comum()

{System.out.println("classe veiculo");}

2 - Insira na interface Bens_De_Consumo a seguinte promessa de implementação:

public void comum();

O código abaixo poderá ser compilado sem erros?

O que resulta da sua execução?

```
1  public class Automovel extends Veiculo implements Bens_de_Consumo {
2      private double Valor;
3
4      public double calculoDepreciacao(int anoFabricacao, double taxa) {
5          java.util.Date hoje = new java.util.Date();
6          int anoHoje = 1900+hoje.getYear();
7          return (Valor*(1-(Math.abs(anoHoje-anoFabricacao)*(1-taxa/100))/100));
8      }
9
10     public void setValor(double v) {Valor=v;}
11
12     public double getValor() {
13         return Valor;
14     }
15
16     public static void main(String [] a)
17     {  Automovel gol = new Automovel();
18        gol.setValor(12000);
19        System.out.println(gol.calculoDepreciacao(1980,2));
20        gol.comum();
21     }
22 }
23
```

18 Pacotes e Classes

Introdução:

A criação de uma aplicação em Java envolve a criação de várias classes. Mesmo que seja usado apenas um único arquivo ".java" contendo os fontes das classes, para cada classe, um arquivo ".class" é gerado para cada classe definida.

Sem um mecanismo de organização, seria necessário descobrir que classes são necessárias para a execução de uma aplicação qualquer, e a falta de uma classe poderia impedir a execução de toda a aplicação.

Com a finalidade de organizar esse conjunto de classes, o Java fornece uma solução. Java provê um mecanismo de agrupamento de classes em pacotes (packages). Nesse pacote, pode-se criar grupos de classes que mantém uma relação entre si. Um pacote pode servir também para indicar quem criou as classes.

Até o presente momento, as classes têm sido criadas sem declarar a que pacote elas pertencem. Todas as classes criadas com essa característica pertencem ao pacote *default*.

Criação de pacotes de classes:

Pacotes requerem que as classes que o compõe sejam armazenadas em um mesmo específico diretório. O diretório necessita possuir o mesmo nome do pacote.

Para definir se uma classe pertence ou não a um pacote, basta declarar no início do arquivo

package <nome do pacote> .

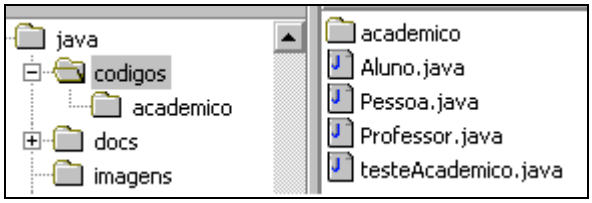
Exemplo:

```
////////////////// arquivo testeAcademico.java
package academico;

import javax.swing.*;
import java.util.*;

public class testeAcademico
{
 public static Professor novoProfessor()
  {
    // o codigo eh omitido por uma questão de espaço.
  }

  public static void main(String args[])
  {
            Professor p1 = novoProfessor();
            System.out.println(p1);
  }
}
```

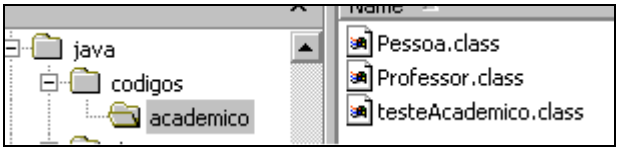
O efeito da definição de pacotes na estrutura de diretório:

A instrução de definição de pacote informa ao compilador Java que as classes devem ser armazenadas em um diretório específico. O Java cria diretórios com o nome do pacote.

Nesse exemplo, os códigos Java estão no diretório “codigos”. A partir desse diretório, o compilador cria a estrutura de diretórios necessária, conforme ilustrado na figura a seguir. Observe que um diretorio chamado “academico” foi criado dentro da pasta codigos.



Dentro desse novo diretório, as classes são depositadas, conforme ilustrado abaixo.



Estrutura de Diretórios:

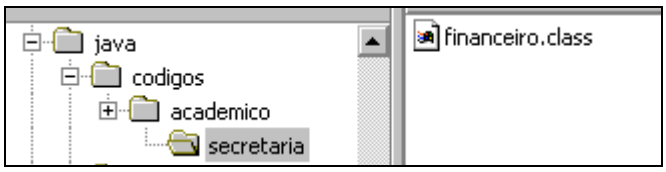
O código a seguir cria uma hierarquia de diretorios ou estruturas na aplicação:

Exemplo:

```
///////////////    arquivo financeiro.java
package academico.secretaria;

public class financeiro {

    public financeiro()
    {    }
}
```

Estrutura do exemplo

O exemplo anterior, se compilado, irá gerar a seguinte estrutura de diretório:



Como utilizar classes de outros pacotes?

Uma classe de um pacote pode utilizar classes de outro pacote. Para acessar o outro pacote, basta inserir a instrução **import <nome do pacote> ;** .

Exemplo:

```
//////////////////// arquivo financeiro.java
package academico.secretaria;
import academico.*;

public class financeiro {

    private Pessoa p;
    private Professor p1;
    public financeiro()
    {    }
}
```

Comentários: No exemplo anterior, as classes Pessoa e Professor foram definidas no pacote academico. Para utilizar essas classes que estão em um pacote diferente do pacote **academico.secretaria** o programador precisa solicitar a sua importação através da instrução **import**.

O modificador de acesso `protected`:

Campos e métodos declarados com esse modificador podem ser usados diretamente por todas as classes pertencentes ao mesmo pacote.

Classes externas ao pacote não podem acessar os campos e os métodos declarados como protegidos.

Exemplo: Na classe Professor, crie um método **print()** com modificador de acesso para **`protected`** conforme a ilustrado abaixo.

```
package academico;

public class Professor extends Pessoa {
    private String disciplina;

    public Professor(String nome) {super(nome

    public void setDisciplina(String materia)

    public String toString() {
     String saida=super.toString();
     saida += "\nDados Acadêmicos do Professc
     saida += "\n\tDisciplina: ";
     saida += getDisciplina();
     return saida;
    }

    public String getDisciplina() {
        return disciplina;
    }

    protected void print()
    {
        System.out.println(this.toString());
    }
}
```

Com essa modificação, a classe financeiro ao tentar acessar o método **`print()`** da classe professor irá fornecer o erro de compilação

ilustrado abaixo.

```
package academico.secretaria;
import academico.*;

public class financeiro {

    private Pessoa p;
    private Professor p1;

    public static void main( String[] a)
    {
        Professor novo = new Professor("exemplo");
        novo.setEmail("teste@teste.com.br");
        novo.print();
    }
}
```

```
Build Output
--------------------Configuration: <Default>--------------------
D:\aulas\mercenario\java\codigos\financeiro.java:13: cannot resolve symbol
symbol  : method print ()
location: class academico.Professor
        novo.print();
             ^
1 error

Process completed.
```

O modificador de acesso protected:

O modificador `protected` fará com que qualquer campo ou método declarado com ele possa ser acessado por qualquer classe que herde da classe.

1. Na classe Pessoa definida no aqruivo Pessoa.java, modifique o acesso do método `getEMail( )` para `protected`. Por exemplo : **`protected String getEMail() {return email;}`**

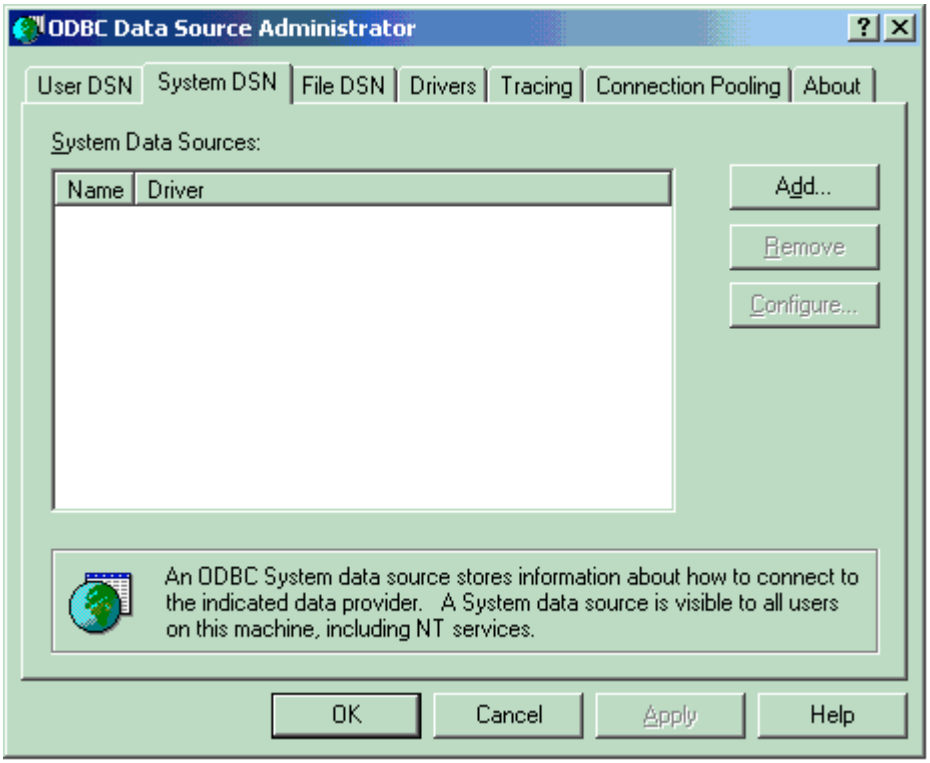
2. Na classe Professor, insira o método conforme destacado no código a seguir.

```
package academico;

public class Professor extends Pessoa {
    private String disciplina;

    public Professor(String nome) {super(nome); }

    public String getEMail() {return super.getEMail();}
```

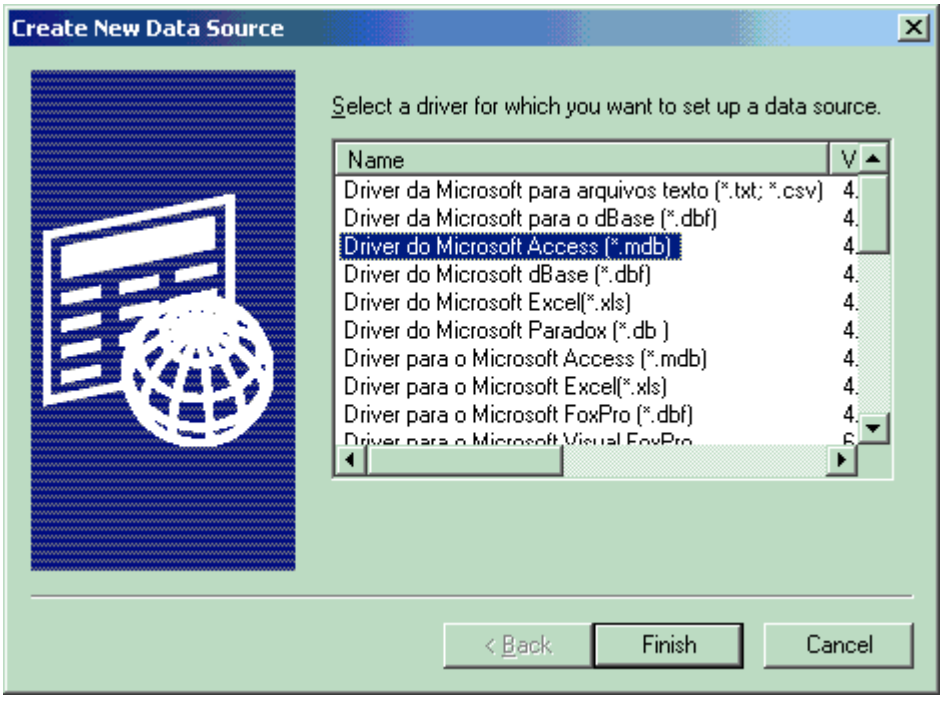
19. Bonus: Como Conectar o Java a um Banco de Dados

Como criar o Banco de Dados MSAccess ?

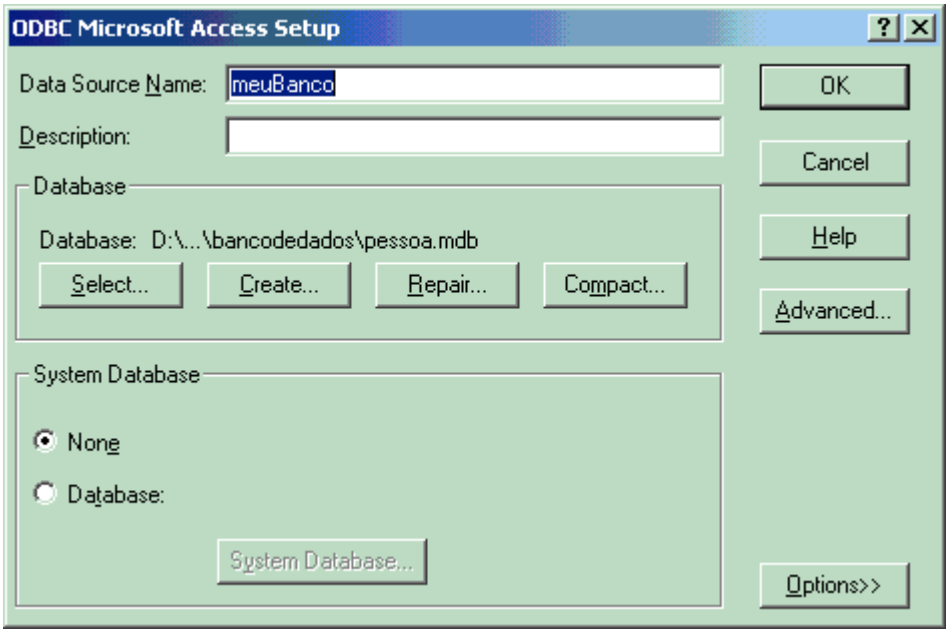
1. Crie um banco de dados no Access e salve em alguma pasta, por exemplo crie uma pasta com o nome “BDCurso” na pasta raiz do seu projeto.
2. Agora abra o Painel de Controle do Windows, vá em Ferramentas Administrativas -> Fonte de Dados (ODBC). Você pode selecionar a aba Fontes de Dados do Sistema, e então clique em Adicionar.



3. Selecione Driver do Microsoft Access (*.mdb) e clique em Concluir,



4. Coloque um nome para o Alias do seu banco
5. Clique em Selecionar e escolha o caminho do banco.



6. Pronto, agora pressione OK e pronto, o acesso ao banco de dados está criado!

Como o programa Java accessa o Banco de Dados ?

1. Nas instruções `import` de um c'odigo java, a seguinte linha precisa ser adicionada: "import java.sql.*; " . Ela é que contém as classes necessárias para se conectar e manipular o banco
2. Agora, dentro da classe, você pode definir alguns objetos, um Connection, que manipula a conexão em si, e um Statement, que manipula o banco com querys.

```
private Connection con;
private Statement stm;
```

3. Crie um método específico para o acesso ao banco.

```
public void open(String ALIAS)
{
 try
 {
  /* Tenta se conectar ao Driver */
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 }
 //tratamento de erro
 catch (ClassNotFoundException e)
 {
  System.out.println("Impossível carregar o Driver: \n" + e);
  System.exit(0);
 }

 try
 {
```

```
    con = DriverManager.getConnection("jdbc:odbc:"+ALIAS);
     /* nesse exemplo ALIAS é o nome do banco definido na
         configuração do ODBC*/
    stm = con.createStatement();
   }
   catch (SQLException e)
   {
    System.out.println( "Não foi possível acessar o banco de dados:\n" +
e);
    System.exit(0);
   }
  }
```

4. Crie um método para encerrar o acesso ao banco.

```
public void close()
{ try{
      if ( ((stm == null) && (con == null)) == false)
       { stm.close(); con.close();}
     }
  catch(java.sql.SQLException e){System.out.println(e);}
}
```

5. O acesso específico à tabela é realizado por uma instância da classe `Statement`. O método `execute(String)` é utlizado para executar uma *query*.

```
stm.execute(
      "insert into nomeDaTabela (nomeDaColuna1, nomeDaColuna2) value (..)");
```

6. Esse método não possui retorno. Caso o resultado ou retorno da instrução SQL seja um conjunto de dados, o método **`executeQuery(String)`** deverá ser utilizado. Esse método retorna um objeto do tipo ResultSet.

```
/*1.*/  try
/*2.*/  {  String dados;
/*3.*/  ResultSet rs = stm.executeQuery(
/*4.*/  "SELECT * FROM nomeDaTabela");
/*5.*/  String dados;
/*6.*/  while (rs.next()) {dados =  rs.getString(0);  }
/*7.*/  }
/*8.*/  catch (SQLException e)
/*9.*/  {System.out.println("Erro: \n " + e);  }
```

Caso seja necessário obter meta-informações da tabela (por exemplo, quantidade de colunas) deve-se utilizar uma instânica da classe **ResultSetMetaData.**

```
ResultSetMetaData metaData = rs.getMetaData();
```

São alguns métodos importantes do ResultSetMetaData:

- `int getColumnCount() retorna a quantidade de colunas.`
- `int getColumnDisplaySize(int column) retorna o tamanho máximo em caracteres definido para a coluna.`
  `String getColumnLabel(int column) retorna o rótulo da coluna.`
- `String getColumnName(int column) retorna o nome da coluna.`
- `int getColumnType(int column) retorna o tipo de dado da coluna.`
- `String getColumnTypeName(int column) retorna o nome do tipo de dado da coluna.`
- `int getPrecision(int column) retorna a quantidade de casas decimais definida para a coluna.`
- `String getSchemaName(int column) retorna o nome do esquema definido para a coluna da tabela.`

Exemplo de Acesso via ODBC:

Nesse exemplo, considera-se a existencia de uma tabela chamada Pessoa para um alias " meuBanco".

```
/*1.*/  // arquivo BancoAccess.java
/*2.*/  import java.sql.*;
/*3.*/  import java.io.*;

/*4.*/  public class BancoAccess
/*5.*/  {
/*6.*/  private Connection con;
/*7.*/  private Statement stm;

/*8.*/  public void open(String ALIAS)
/*9.*/  {
/*10.*/ try
/*11.*/ {
/*12.*/ /* Tenta se conectar ao Driver */
```

```
/*13.*/ Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
/*14.*/ }
/*15.*/ catch (ClassNotFoundException e)
/*16.*/ {
/*17.*/ System.out.println("Impossível carregar o Driver: \n" +
   e);
/*18.*/ System.exit(0);
/*19.*/ }
/*20.*/ try
/*21.*/ {
/*22.*/ con = DriverManager.getConnection("jdbc:odbc:"+ALIAS);
/*23.*/ /* nesse exemplo ALIAS é o nome do banco definido na
   configuração do ODBC*/
/*24.*/ stm = con.createStatement();
/*25.*/ }
/*26.*/ catch (SQLException e)
/*27.*/ {
/*28.*/ System.out.println( "Não foi possível acessar o banco de
   dados:\n" + e);
/*29.*/ System.exit(0);
/*30.*/ }
/*31.*/ }
/*32.*/ public void close()
/*33.*/ { try{
/*34.*/ if ( ((stm == null) && (con == null)) == false)
/*35.*/ { stm.close(); con.close();}
/*36.*/ }
/*37.*/ catch(java.sql.SQLException e){System.out.println(e);}
/*38.*/ }
/*39.*/ public void printResultSet(PrintStream p, ResultSet rs
/*40.*/        ,  String title)
/*41.*/ {
/*42.*/  try{
/*43.*/  if(rs != null)
/*44.*/   { ResultSetMetaData metaData = rs.getMetaData();
/*45.*/     int cols = metaData.getColumnCount();
/*46.*/     p.println("\n-------------------------\n" + title
/*47.*/            + "\n-------------------------");
/*48.*/     for(int i = 1;i <= cols;i++)
```

```
/*49.*/      { p.print(metaData.getColumnLabel(i) + "\t"); }
/*50.*/        p.println("\n-------------------------");
/*51.*/        int count = 0;
/*52.*/        while(rs.next())
/*53.*/       {for(int i = 1;i <= cols;i++)
/*54.*/         { p.print(rs.getString(i) + "\t"); }
/*55.*/           p.println("\n-------------------------");
/*56.*/           count++;
/*57.*/          }
/*58.*/     p.println("Quantidade de Registros: " + count);
/*59.*/   }
/*60.*/  }
/*61.*/  catch (java.sql.SQLException e) {System.out.println(e);}
/*62.*/ }
/*63.*/ public void executeQuery(String pesquisa)
/*64.*/ {
/*65.*/  try { ResultSet rs = stm.executeQuery(pesquisa);
/*66.*/        printResultSet(System.out, rs
/*67.*/                    ,"Dados Armazenados na Tabela");
/*68.*/      }
/*69.*/  catch (java.sql.SQLException e) {System.out.println(e);}
/*70.*/ }
/*71.*/ public void execute(String comando)
/*72.*/ { try { stm.execute(comando); }
/*73.*/   catch (java.sql.SQLException e) {System.out.println(e);}
/*74.*/ }
/*75.*/  public static void main(String [] a)
/*76.*/  {
/*77.*/   BancoAccess banco = new BancoAccess();
/*78.*/   banco.open("meuBanco");
/*79.*/   banco.executeQuery("select * from pessoa");
/*80.*/   banco.close();
/*81.*/  }}
```

Exemplo de Acesso a um Banco de Dados MySql através de drive JDBC.

```
                                                  // arquivo Conectar.java
/*1.*/  import java.sql.*;
/*2.*/
/*3.*/  public class Conectar
```

```
/*4.*/  {
/*5.*/    private  Connection con;
/*6.*/    private  Statement stm;
/*7.*/    private  String bd= "jdbc:mysql://localhost:3306/";
/*8.*/    private  String driver="com.mysql.jdbc.Driver";
/*9.*/    private  String usuario="root";
/*10.*/   private  String senha="root";
/*11.*/   private  String dataBase="exemplo";
/*12.*/
/*13.*/   Conectar(String aonde)
/*14.*/   { bd = aonde;  }
/*15.*/   Conectar(){}
/*16.*/   public void setDataBase(String v){dataBase=v;}
/*17.*/   public String getDataBase(){return dataBase;}
/*18.*/
/*19.*/   public  void open()
/*20.*/   {
/*21.*/   try
/*22.*/    {Class.forName(driver);
/*23.*/     con = DriverManager.getConnection(bd,usuario,senha);
/*24.*/     stm = con.createStatement();
/*25.*/     }
/*26.*/    catch(java.lang.Exception e) {System.out.println(e);}
/*27.*/    }
/*28.*/  public  void close()
/*29.*/  { try{ if (closed()==false)
/*30.*/        { stm.close(); con.close();}
/*31.*/       }
/*32.*/  catch(java.sql.SQLException e){System.out.println(e);}
/*33.*/  }
/*34.*/  public  boolean closed()
/*35.*/  { return ((stm == null) && (con == null)); }
/*36.*/  public  ResultSet executeQuery(String sql)
/*37.*/  {  try{ if (closed() ) open();
/*38.*/           return stm.executeQuery(sql);
/*39.*/         }
/*40.*/   catch(java.sql.SQLException e)
/*41.*/             {System.out.println(e); return null;}
/*42.*/  }
```

```
/*43.*/
/*44.*/  public  void setBD(String v){bd=v;}
/*45.*/  public  String getBD(){return bd;}
/*46.*/
/*47.*/  public  void executeInsert(int codigo , String nome
/*48.*/                       , double valor)
/*49.*/  {
/*50.*/   String comando="insert "+dataBase
              +".produto (codigo, nome,valor) values ("
              + codigo
              + ", \'" + nome + "\'"
              + "," + valor + ")";
/*51.*/    executeMe(comando);
/*52.*/  }
/*53.*/  public  void executeUpdate(int codigo
/*54.*/          , String nome, double valor)
/*55.*/  {
/*56.*/   String comando=
/*57.*/  "update "+dataBase+".produto set nome=\'"
                + nome + "\'"
                + ", valor=" + valor
                + " where codigo=" + codigo;
/*58.*/    executeMe(comando);
/*59.*/  }
/*60.*/  public  void delete(int codigo)
/*61.*/  {
/*62.*/  String comando="delete from "
            +dataBase+".produto where codigo="
            + codigo;
/*63.*/  executeMe(comando);
/*64.*/  }
/*65.*/  public  void executeMe(String comando)
/*66.*/  {
/*67.*/  try
/*68.*/  {
/*69.*/     if (closed()) open();
/*70.*/     stm.executeUpdate(comando);
/*71.*/   } catch(java.lang.Exception e){System.out.println(e);}
/*72.*/  }
```

```
/*73.*/  public  void criarTabela()
/*74.*/  {
/*75.*/   String comando;
/*76.*/   comando = "create table if not exists "
                + dataBase+".produto "
                + "(codigo int unsigned not null auto_increment,"
                + " nome varchar(100) not null,"
                + " valor double ,"
                + " primary key(codigo)"
                + ") engine=innodb default charset=latin1;";
/*77.*/    executeMe("drop table if exists "
                        +dataBase+".produto");
/*78.*/    executeMe(comando);
/*79.*/  }
/*80.*/  public  void criarDataBase(String dataBase)
/*81.*/  { setDataBase(dataBase);
/*82.*/    String comando = "CREATE SCHEMA IF NOT EXISTS "
                              +dataBase;
/*83.*/     try {if (closed() ) open();
/*84.*/          stm.executeUpdate("DROP SCHEMA "+dataBase);
/*85.*/         } catch(java.lang.Exception e){}
/*86.*/
/*87.*/    executeMe(comando);
/*88.*/  }
/*89.*/ }
```

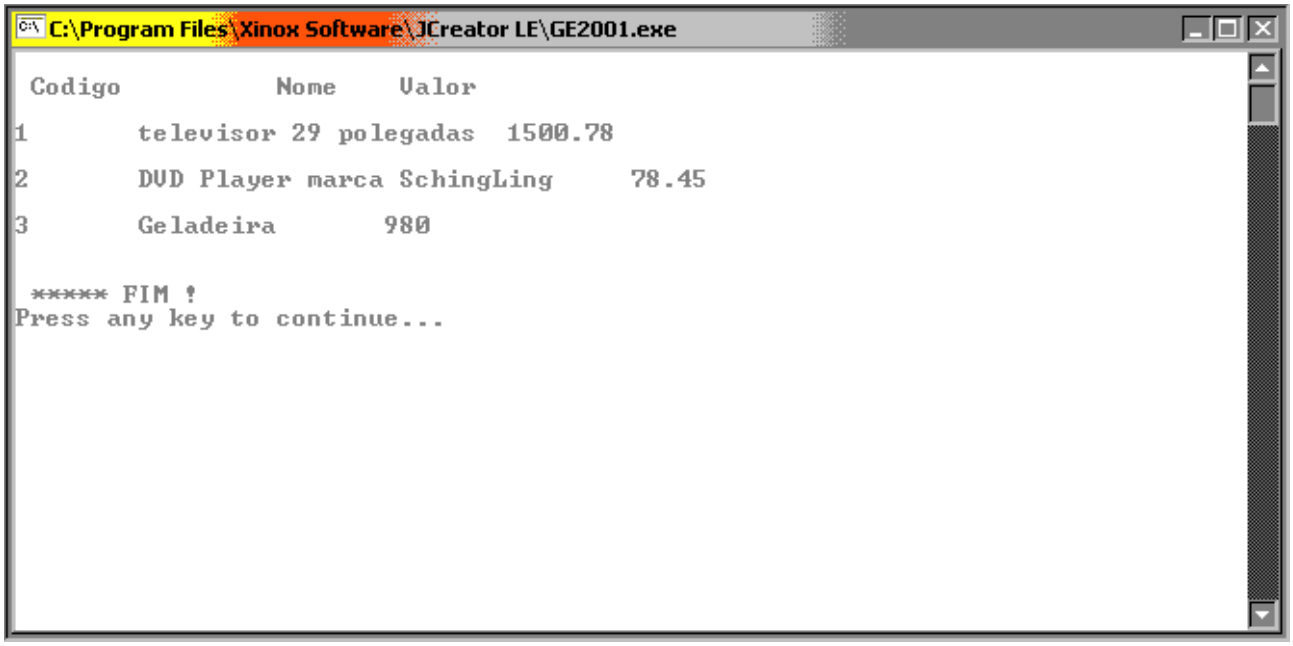
Exemplo do uso da classe.

O código a seguir apresenta um exemplo de execução da classe Conectar.

```
/*1.*/  import java.sql.*;
/*2.*/  class BancoDeDados
/*3.*/  {
/*4.*/   public static void main(String[] a)
/*5.*/   {
/*6.*/    Conectar banco = new
   Conectar("jdbc:mysql://localhost:3306/");
/*7.*/    banco.criarDataBase("lojinha");
/*8.*/    banco.criarTabela();
/*9.*/    banco.executeInsert(1,"televisor 29 polegadas",1500.78);
/*10.*/   banco.executeInsert(2,"DVD Player marca SchingLing"
                         , 78.45);
/*11.*/   banco.executeInsert(3,"Geladeira ", 980.00);
```

```
/*12.*/
/*13.*/   ResultSet rec;
/*14.*/   rec = banco.executeQuery(
                        "select * from lojinha.produto");
/*15.*/
/*16.*/   System.out.println("\n Codigo \t Nome \t Valor ");
/*17.*/   try
/*18.*/   {
/*19.*/    while(rec.next())
/*20.*/    {
/*21.*/    System.out.println("\n" + rec.getString("codigo")
/*22.*/  + "\t" + rec.getString(2)
/*23.*/  + "\t" + rec.getString(3)
/*24.*/  + "\t");
/*25.*/  }
/*26.*/  }
/*27.*/  catch(java.sql.SQLException e){System.out.println(e);}
/*28.*/   System.out.println("\n\n ***** FIM !");
/*29.*/ }}
```

A execução desse código deverá fornecer a seguinte saída:



Por outro lado, a execução desse código no ambiente do JCreator, pode fornecer a seguinte mensagem de erro:

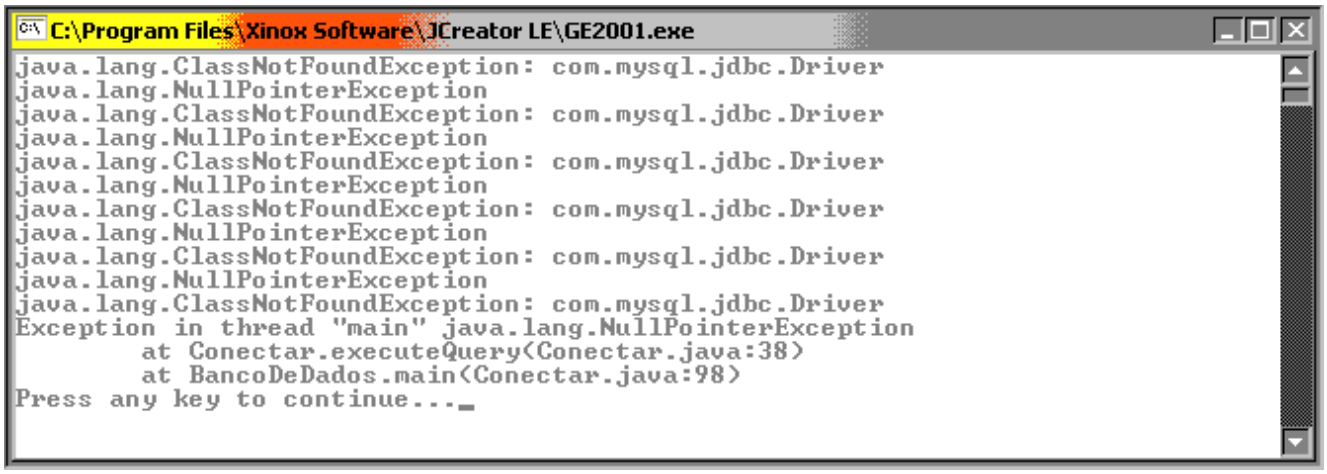


Figura 8- Possível erro de execução da classe Conectar

Esse erro ocorre pela ausência do `driver` de acesso ao MySql. Nesse caso, deve-se adicionar o driver (mysql-connector-java-5.1.7-bin.jar) ao ambiente. Como fazer isso?

1. Selecione o menu Configure | Options.
2. Na janela que aparecerá, selecione JDK Profiles.

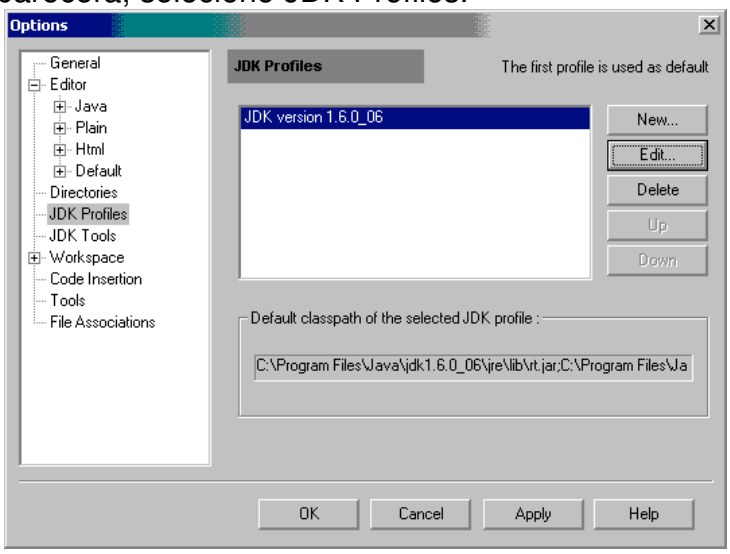
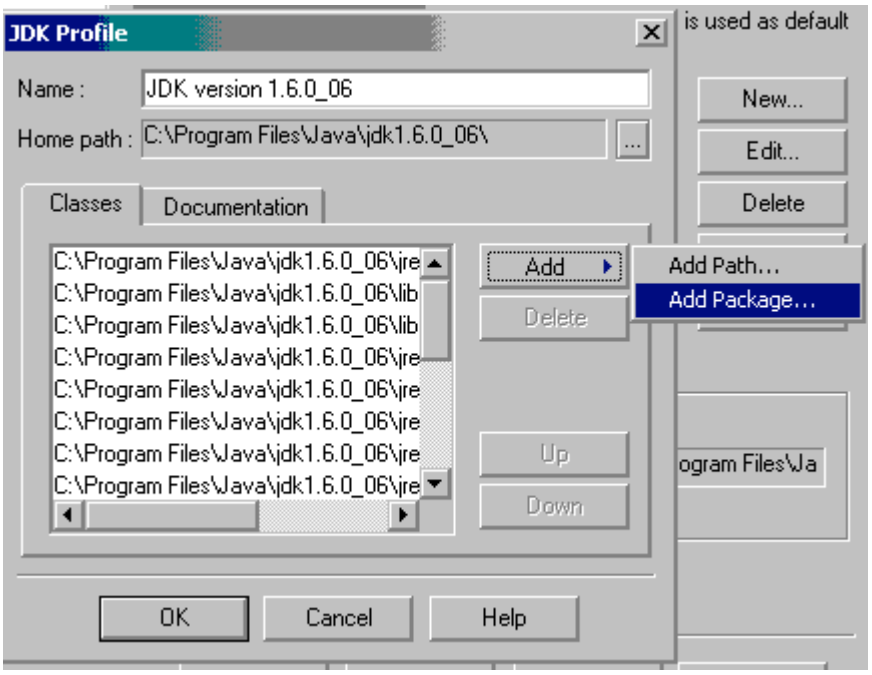


Figura 9-Janela de configuração do JCreator

3. Clique no botão Edit.
4. Na janela que aparecerá, selecione o botão Add | Add Packages.



5. Uma cópia do driver está na pasta java|mysql. Indique esse caminho ou copie esse driver para uma pasta de seu computador e aponte para esta pasta. Se voce for copiar esse arquivo para dentro do seu computador, recomendo que essa pasta esteja dentro da pasta de instalação do Java.

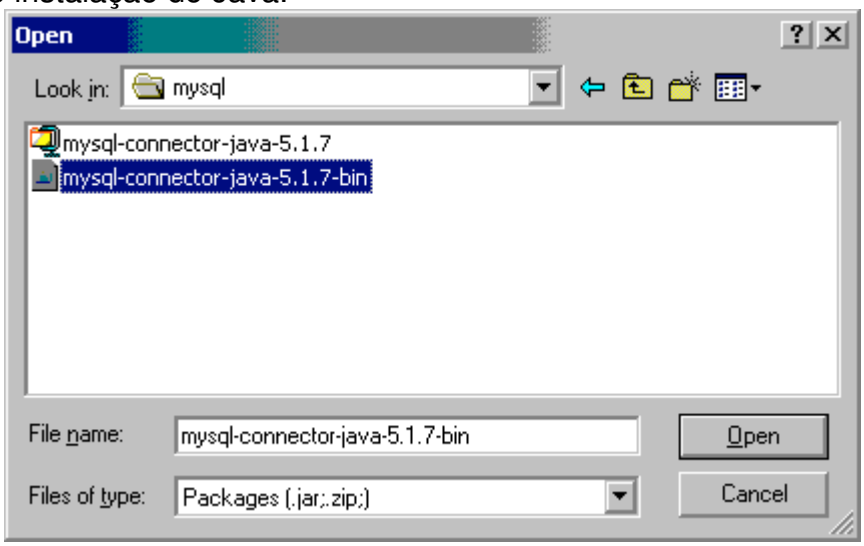


Figura 10-Arquivo de Driver mysql

6. Em seguida, clique nos botões OK e Apply para encerrar a configuração.

20. Bonus 2: Empacotamento de Arquivos para Distribuição (“.jar”)

A distribuição de aplicativos Java para o usuário final poderia ser realizada de diversas maneiras. Esta seção apresenta apenas uma delas. Nessa apresentação, toda a aplicação é empacotada em um arquivo **`.jar`**. Esse é o único arquivo que precisará ser enviado ao usuário.

Inicialmente, iremos preparar um simples exemplo para compreendermos a dinâmica do processor. Nos dois quadros a seguir, um exemplo de aplicação é sugerido. O objetivo desta seção é mostrar como essa aplicação pode ser preparada para ser enviada ao usuário.

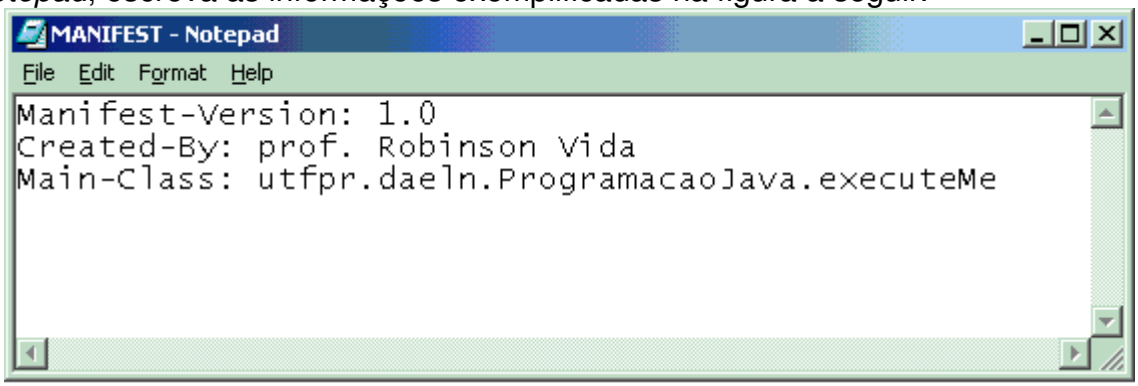
```
package utfpr.daeln.ProgramacaoJava;

class ExemploSimples
{     private int numero;
      public void setNumero(int x) {numero=x;}
      public int getNumero() { return numero;}
}
```

```
package utfpr.daeln.ProgramacaoJava;
import java.util.Date;

class executeMe
{
      public static void main(String[] a)
      { ExemploSimples teste = new ExemploSimples();
        teste.setNumero(5);
        Date hoje = new Date();
        System.out.println(teste.getNumero() + "\t" + hoje);
      }
}
```

Após a compilação da aplicação, ou seja, os arquivos **`.class`** de cada uma das duas classes necessárias deverá estar pronto, iremos preparar a aplicação para a entrega ao usuário final.

- Passos a serem seguidos:

1) Criar um arquivo `MANIFESTO.MF`. Um arquivo manifesto é um arquivo texto que registra algumas importantes informações a respeito da aplicação.Com o editor de notas ou *notepad*, escreva as informações exemplificadas na figura a seguir.

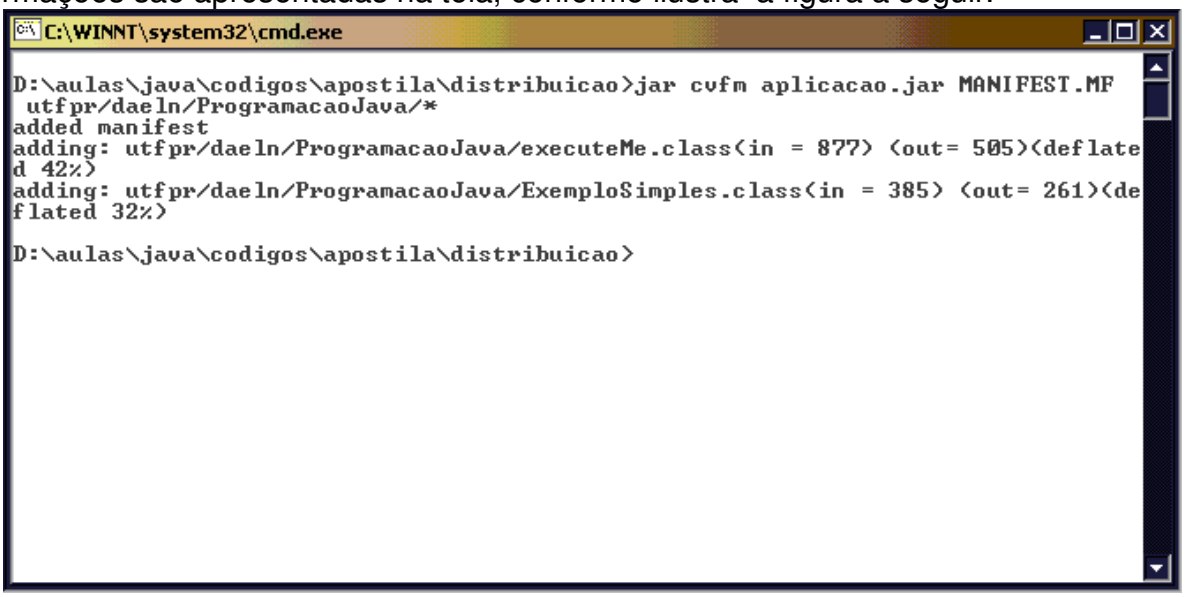


2) Para criar o arquivo **`.jar`** que será entregue ao usuário, o aplicativo ***`jar`*** deverá ser utilizado. Para isso, a seguinte instrução deverá ser DIGITADA:

jar cvfm <nome do arqui.jar> MANIFEST.MF <diretorio onde as classes estao armazenadas>

A figura a seguir ilustra esse procedimento.

3) O resultado será a criação de um arquivo **.jar**. Durante a execução desse, algumas informações são apresentadas na tela, conforme ilustra a figura a seguir.



O arquivo `.jar` criado já pode ser distribuído ao usuário final.
Como esse usuário irá executar essa aplicação? Ele necessitará chamar o interpretador java informando que se trata de uma aplicação jar. Por exemplo:

```
java -jar <nome do arquivo>.jar
```

A figura a seguir ilustra a execução da aplicação.

Finalizando, uma aplicação java deverá ser entregue ao usuário empacotada, ou seja, no formato `.jar`.

Resumo de alguns métodos da classe String

Method Summary	
char	**charAt**(int index) Returns the character at the specified index.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contentEquals**(StringBuffer sb) Returns `true` if and only if this `String` represents the same sequence of characters as the specified `StringBuffer`.
boolean	**endsWith**(String suffix) Tests if this string ends with the specified suffix.
boolean	**equals**(Object anObject) Compares this string to the specified object.
boolean	**equalsIgnoreCase**(String anotherString) Compares this `String` to another `String`, ignoring case considerations.
void	**getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin) Copies characters from this string into the destination character array.
int	**indexOf**(int ch) Returns the index within this string of the first occurrence of the specified character.
int	**indexOf**(int ch, int fromIndex) Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
int	**indexOf**(String str) Returns the index within this string of the first occurrence of the specified substring.
int	**indexOf**(String str, int fromIndex) Returns the index within this string of the first occurrence of the specified

	substring, starting at the specified index.
int	**lastIndexOf**(int ch) Returns the index within this string of the last occurrence of the specified character.
int	**lastIndexOf**(int ch, int fromIndex) Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.
int	**lastIndexOf**(String str) Returns the index within this string of the rightmost occurrence of the specified substring.
int	**lastIndexOf**(String str, int fromIndex) Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.
int	**length**() Returns the length of this string.
boolean	**matches**(String regex) Tells whether or not this string matches the given regular expression.
boolean	**regionMatches**(boolean ignoreCase, int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
boolean	**regionMatches**(int toffset, String other, int ooffset, int len) Tests if two string regions are equal.
String	**replace**(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
String	**replaceAll**(String regex, String replacement) Replaces each substring of this string that matches the given regular expression with the given replacement.
String	**replaceFirst**(String regex, String replacement) Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[]	**split**(String regex) Splits this string around matches of the given regular expression.
String[]	**split**(String regex, int limit) Splits this string around matches of the given regular expression.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if this string starts with the specified prefix beginning a specified index.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.
String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
char[]	**toCharArray**() Converts this string to a new character array.
String	**toLowerCase**() Converts all of the characters in this String to lower case using the rules of the default locale.
String	**toLowerCase**(Locale locale) Converts all of the characters in this String to lower case using the rules of the given Locale.
String	**toString**()

	This object (which is already a string!) is itself returned.
String	**toUpperCase**() Converts all of the characters in this String to upper case using the rules of the default locale.
String	**toUpperCase**(Locale locale) Converts all of the characters in this String to upper case using the rules of the given Locale.
String	**trim**() Returns a copy of the string, with leading and trailing whitespace omitted.
static String	**valueOf**(boolean b) Returns the string representation of the boolean argument.
static String	**valueOf**(char c) Returns the string representation of the char argument.
static String	**valueOf**(char[] data) Returns the string representation of the char array argument.
static String	**valueOf**(char[] data, int offset, int count) Returns the string representation of a specific subarray of the char array argument.
static String	**valueOf**(double d) Returns the string representation of the double argument.
static String	**valueOf**(float f) Returns the string representation of the float argument.
static String	**valueOf**(int i) Returns the string representation of the int argument.
static String	**valueOf**(long l) Returns the string representation of the long argument.
static String	**valueOf**(Object obj) Returns the string representation of the Object argument.

Resumo de alguns métodos da classe Math

Field Summary	
static double	**E** The double value that is closer than any other to *e*, the base of the natural logarithms.
static double	**PI** The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

Method Summary	
static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of an angle, in the range of 0.0 through *pi*.
static	**asin**(double a)

double	Returns the arc sine of an angle, in the range of -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of an angle, in the range of -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Converts rectangular coordinates (x, y) to polar (r, *theta*).
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is not less than the argument and is equal to a mathematical integer.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**exp**(double a) Returns Euler's number *e* raised to the power of a double value.
static double	**floor**(double a) Returns the largest (closest to positive infinity) double value that is not greater than the argument and is equal to a mathematical integer.
static double	**IEEEremainder**(double f1, double f2) Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
static double	**log**(double a) Returns the natural logarithm (base *e*) of a double value.
static double	**max**(double a, double b) Returns the greater of two double values.
static float	**max**(float a, float b) Returns the greater of two float values.
static int	**max**(int a, int b) Returns the greater of two int values.
static long	**max**(long a, long b) Returns the greater of two long values.
static double	**min**(double a, double b) Returns the smaller of two double values.
static float	**min**(float a, float b) Returns the smaller of two float values.
static int	**min**(int a, int b) Returns the smaller of two int values.
static long	**min**(long a, long b) Returns the smaller of two long values.
static double	**pow**(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	**random**() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	**rint**(double a) Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	**round**(double a) Returns the closest long to the argument.
static int	**round**(float a) Returns the closest int to the argument.
static double	**sin**(double a) Returns the trigonometric sine of an angle.
static	**sqrt**(double a)

double	Returns the correctly rounded positive square root of a `double` value.
static double	**tan**(double a) Returns the trigonometric tangent of an angle.
static double	**toDegrees**(double angrad) Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
static double	**toRadians**(double angdeg) Converts an angle measured in degrees to an approximately equivalent angle measured in radians.