EAD

EDUCAÇÃO A DISTÂNCIA

UNIDADE III

Linguagem de Programação Orientada a Objetos

Prof. Me. Ricardo Veras

Polimorfismo

- O Polimorfismo, no Java, é a capacidade de um determinado elemento se comportar de formas diferentes, dependendo de como este elemento é declarado, ou como está sendo utilizado, a partir da sua codificação.

Vimos, por exemplo, que uma variável pode se comportar como uma variável simples ou como um atributo:

```
public void setIdade(int idade) {
    this.idade = idade;
}
```

- Este tipo de Polimorfismo está relacionado à variável.

Polimorfismo

- Vimos também dois tipos de Polimorfismo de métodos: a Sobrecarga e a Sobrescrita.
  - Sobrecarga: métodos com nomes iguais, mas assinaturas diferentes: diferenciados nos parâmetros (na quantidade ou na tipagem). Este tipo de polimorfismo pode ocorrer em uma mesma Classe ou entre Classes Mãe e Filha (herança).
  - Sobrescrita: métodos com assinaturas iguais. Este tipo de polimorfismo não pode ocorrer em uma mesma Classe. No entanto ele ocorre entre Classes Mãe e Filha, ou seja, se houver herança de Classes. Uma Classe Filha possui os métodos da Classe Mãe. Neste caso, havendo na 1ª um método com mesma assinatura de outro método existente na Classe Mãe, este último está sendo substituído pelo outro.

Polimorfismo

Polimorfismo de Classes:

- Em O.O., uma Classe Mãe pode se comportar como uma de suas Classes Filhas (a Classe que herda). Este tipo de polimorfismo aparece em métodos que possuem um parâmetro que hora recebe um tipo de dado (um objeto), hora recebe outro tipo de dado (outro objeto – diferente), mas cuja lógica geralmente é igual para todos os casos:

```
public void actionPerformed(ActionEvent e) {
   Object o = e.getSource();
   ...
}
```

  - Neste caso, o objeto “o” declarado como Object (que é a Classe Mãe de todas as Classes) vai se comportar como os diversos elementos de um Formulário, dependendo de qual foi acionado.

Polimorfismo

Exemplo de Polimorfismo de Classes:

```
public class Carro {
    public void andarParaFrente(Carro crr) {
       //código que atualiza a pista quando o carro anda para frente
       //além de verificar a posição do carro na pista
    }
}
```

```
public class Fusca extends Carro {
     // características do Fusca
}
public class Corola extends Carro {
     // características do Fusca
}
```

Polimorfismo

Exemplo de Polimorfismo de Classes (continuação):

```
public class Corrida {
    public void iniciarCorrida() {
        Fusca f1 = new Fusca();
        Corola c1 = new Corola();
        f1.andarParaFrente(f1);
        c1.andarParaFrente(c1);
        ...
    }
    ...
}
```

Modificadores de Comportamento

- Os Modificadores de Comportamento são modificadores “somados” aos modificadores de acesso e que alteram o comportamento dos Elementos (Classes, métodos e atributos).

São 3 os Modificadores de Comportamento:

- Static
- Final
- Abstract

A sintaxe na Declaração do Elemento deve ser:

```
modifAcesso modifComport declaracaoDoElemento
```

Exemplo:

```
public abstract class ClasseA {...}
public final void metodo01() {...}
```

Modificadores de Comportamento

Static

- O modificador *static* pode ser usado em métodos ou atributos e é utilizado para criar um atributo ou um método que pode ser acessado sem precisar instanciar a Classe a que pertence, ou seja, sem a necessidade de se gerar o objeto para poder chamá-los.

Exemplo:

a Classe “Math”, em que seus métodos são estáticos:

```
x = Math.cos(3.45);
```

Final

- O modificador *final* pode ser usado em Classes, métodos e atributos. Quando declaramos uma Classe como final, esta não poderá ser herdada. Quando declaramos um método como final, este não poderá ser reescrito (ou sobrescrito) em uma subclasse. Quando declaramos um atributo como final, seu valor não poderá ser alterado (vira uma constante).
- Obs.: um atributo final – com nome inteiramente em “caixa-alta” (letras maiúsculas), que é a convenção para constantes.

Modificadores de Comportamento

Static

Exemplo – tem-se as Classes ClasseA e ClasseB abaixo:

```
public class ClasseA {
    public static final double TAXA_X = 0.2763;
    public static String metodo01(String txt) {
        ...
    }
}
```

```
public class ClasseB {
    public double taxa_z = 0.1572;
    public String metodo02(String txt) {
        ...
    }
}
```

Modificadores de Comportamento

Static

Exemplo – continuação:

Suponha uma Classe Teste que vá se utilizar destas duas Classes do exemplo:

```
public class Teste {
    //para utilizar os elementos da Classe A, basta chama-los diretamente
    double d1 = ClasseA.TAXA_X;
    String s1 = ClasseA.metodo01(“texto inicial”)
    //para utilizar os elementos da Classe B, é necessário instanciá-la antes
                        ClasseB cb = new ClasseB();
                        double d2 = cb.taxa_z;
                        String s2 = cb.metodo02(“texto inicial”)
                    }
```

Modificadores de Comportamento

Final

Exemplo – tem-se as Classes ClasseA e ClasseB abaixo:

```
public class ClasseA {
    public final String metodo01(String txt) {
        ...
    }
}
```

```
public class ClasseB extends ClasseA {
    public String metodo01(String txt) { //ERRO
        ...
    }
}
```

Modificadores de Comportamento

Final

Exemplo – tem-se as Classes ClasseA e ClasseB abaixo:

```
public final class ClasseA {
    public String metodo01(String txt) {
        ...
    }
}
```

```
public class ClasseB extends ClasseA { //ERRO
     ...
}
```

Interatividade

Durante o processo de implementação de um sistema Java instalado num cliente, mesmo sem conhecer exatamente como o mesmo foi construído, qual das opções abaixo mostra o que é possível deduzir apenas verificando-se as seguintes linhas de código, localizadas “dentro” de um método de uma determinada classe do sistema?

```
...
String codigoProduto;
//... mais linhas de código
double taxa = Produto.verifica_ICMS(codigoProduto);
```

a) Que o Código do Produto é um atributo da classe Produto.
b) Que o método “verifica_ICMS(...)” é um método estático e que pertence a uma classe chamada Produto.
c) Que o método “verifica_ICMS(...)” pode não possuir retorno de informação.
d) Que a variável codigoProduto é uma constante.
e) Que “Produto” é um objeto instanciado.

Resposta

Durante o processo de implementação de um sistema Java instalado num cliente, mesmo sem conhecer exatamente como o mesmo foi construído, qual das opções abaixo mostra o que é possível deduzir apenas verificando-se as seguintes linhas de código, localizadas “dentro” de um método de uma determinada classe do sistema?

```
...
String codigoProduto;
//... mais linhas de código
double taxa = Produto.verifica_ICMS(codigoProduto);
```

a) Que o Código do Produto é um atributo da classe Produto.
b) Que o método “verifica_ICMS(...)” é um método estático e que pertence a uma classe chamada Produto.
c) Que o método “verifica_ICMS(...)” pode não possuir retorno de informação.
d) Que a variável codigoProduto é uma constante.
e) Que “Produto” é um objeto instanciado.

Modificadores de Comportamento

O modificador “abstract”

Este modificador pode ser usado em Classes ou métodos.

- As Classes Abstratas serão Classes que não podem ser instanciadas (são utilizadas com elementos estáticos ou a partir das suas Classes Filhas).
- Métodos Abstratos não possuem implementação na Classe em que se localizam. Mas deverão ser implementados nas Classes Filhas.

Modificadores de Comportamento

O modificador “abstract” (continuação)

- Para as Classes, este modificador apenas impede que ela seja instanciada (não pode ser gerado um objeto com esta Classe), o que quer dizer que os elementos desta Classe (seus atributos e métodos) somente serão utilizados através de suas Classes Filhas (lembrando que quando uma Classe herda outra Classe, a “Classe Filha” adquire todos os elementos da “Classe Mãe").

Modificadores de Comportamento

O modificador “abstract” (continuação)

Exemplo:

```
public abstract class Carro {
	//...Elementos da Classe Carro
}
public class Fusca extends Carro {
	//...Elementos da Classe Fusca
}
```

Assim, numa outra classe, podemos instanciar a Classe Fusca, mas não a Classe Carro:

```
...
Fusca f1 = new Fusca();
...
```

- ...de forma que a partir do objeto “f1”, podemos acessar os elementos da classe Carro, já que Fusca herdou esses elementos.

Modificadores de Comportamento

O modificador “abstract” (continuação)

Para os métodos, este modificador determina algumas características:

- Métodos abstratos só podem existir em Classes abstratas;
- Métodos abstratos não possuem implementação;
  - `public abstract tipoRetorno nomeMetodo(parâmetros);`
  - (percebe-se que métodos abstratos terminam com “ponto-e-vírgula”)
- Os métodos abstratos necessitam obrigatoriamente ser implementados nas Classes Filhas (classes filhas da Classe em que aquele método foi declarado).

Modificadores de Comportamento

O modificador “abstract” (continuação)

Observações importantes:

- Sabemos que Métodos abstratos só podem existir em Classes abstratas;
- Mas Classes abstratas podem sim ter métodos comuns e/ou também métodos abstratos (elas não precisam possuir métodos abstratos);
- (classes abstratas são, simplesmente, Classes que não podem ser instanciadas)
- Se uma Classe abstrata herda outra Classe abstrata, ela não precisa implementar os métodos abstratos da Classe herdada (caso existam).

Modificadores de Comportamento

O modificador “abstract” (exemplo)

```
public abstract class ClasseA {
	public abstract void metodo01(String txt);
	//...demais elementos da Classe ClasseA
}
public class Teste {
	ClasseA ca = new ClasseA(); //ERRO
	...
}
```

Modificadores de Comportamento

O modificador “abstract” (exemplo)

```
public abstract class ClasseA {
	public abstract void metodo01(String txt);
	//...demais elementos da Classe ClasseA
}
public class ClasseB extends ClasseA {
	public void metodo01(String txt) {  // ..obrigatório
		...
	}
		...
		}
```

Modificadores de Comportamento

Tabela Resumo:

	static	final	abstract
Classe		**Não** pode ser herdada por outra Classe.	**Não** pode ser instanciada (é utilizada através de suas Classes Filhas).
método	Pode ser acionado **sem** precisar instanciar a Classe a que pertence.	**Não** pode ser sobrescrito (não pode ser substituído por outro método em uma SubClasse).	• É um método sem implementação (só com a declaração). • **Deverá ser** implementado em uma Classe Filha. • **Somente** pode existir em uma Classe Abstrata.
atributo	Pode ser lido **sem** precisar instanciar a Classe a que pertence (deve possuir valor na declaração).	São **constantes** (seu nome deve estar em Caixa-Alta).	

Tabela demonstrativa dos tipos de modificadores de comportamento.
Fonte: própria

Interatividade

Tem-se uma classe ClasseA abstrata com 5 métodos públicos nela implementados, de forma que não há método abstrato algum. Seja uma ClasseB final, que herda aquela ClasseA. De acordo com este cenário, selecione abaixo uma opção correta.

a) É impossível não haver método abstrato algum na ClasseA, já que toda classe abstrata deve conter ao menos um método abstrato.

b) A ClasseB, por ser “final”, não poderia herdar a ClasseA.

c) A ClasseA, por ser “abstrata”, não poderia ser herdada por nenhuma outra classe.

d) Os métodos da ClasseA poderão ser acionados a partir de instâncias (objetos) geradas a partir da ClasseB.

e) Na ClasseB deverá haver obrigatoriamente uma implementação de todos os métodos da ClasseA, sobrescrevendo-os.

Resposta

Tem-se uma classe ClasseA abstrata com 5 métodos públicos nela implementados, de forma que não há método abstrato algum. Seja uma ClasseB final, que herda aquela ClasseA. De acordo com este cenário, selecione abaixo uma opção correta.

a) É impossível não haver método abstrato algum na ClasseA, já que toda classe abstrata deve conter ao menos um método abstrato.

b) A ClasseB, por ser “final”, não poderia herdar a ClasseA.

c) A ClasseA, por ser “abstrata”, não poderia ser herdada por nenhuma outra classe.

d) Os métodos da ClasseA poderão ser acionados a partir de instâncias (objetos) geradas a partir da ClasseB.

e) Na ClasseB deverá haver obrigatoriamente uma implementação de todos os métodos da ClasseA, sobrescrevendo-os.

Interface

- Uma Interface é um elemento que equivale a uma Classe Abstrata, mas que só pode possuir métodos abstratos, e/ou atributos finais.

```
public interface InterfaceA {
    ...
}
```

- Uma interface é implementada por uma Classe através da palavra-chave implements, ao final da declaração da Classe.

```
public class ClasseA implements InterfaceA {
    ...
}
```

Interface

- O principal objetivo de uma interface é obrigar as Classes que a implementarem, a construírem (implementarem) os métodos que a interface possui.

Algumas características da interface:

- Uma interface somente possuirá métodos abstratos e/ou atributos finais (constantes).
- O nome de uma interface segue o mesmo padrão de nomes de Classes (começa com maiúsculo).
- Uma Classe pode implementar múltiplas interfaces:

```
public class ClsA implements InterfA, InterfB, InterfC {
      ...
}
```

  - ...Neste caso, ela terá que implementar todos os métodos de todas as interfaces que ela implementou.

Interface

Observações importantes:

- Os métodos de uma interface não levam o modificador abstract (apesar de serem abstract);
- Os atributos de uma interface não levam o modificador final (apesar de serem final);

Uma explicação “prática” para Interfaces:

- Quando uma Empresa possui uma ou mais regras que devem ser seguidas e colocadas corretamente em um programa O.O., então sempre que uma Classe que vai representar um produto que precisa seguir aquelas regras for criada, essas regras deverão ser implementadas na Classe.

  Uma das formas de garantir que isso seja feito, é criar uma Interface contendo a declaração dessas regras, o que no mínimo vai obrigar ao Desenvolvedor conhecer o produto no qual ele está trabalhando e procurar conhecer suas particularidades, para depois gerar suas representações em Java.

A Classe JOptionPane

- JOptionPane é uma classe que possibilita a criação de uma “caixa de dialogo” padrão que ou solicita um valor para o usuário, ou mostra alguma informação.

Métodos	Descrição
showConfirmDialog	Solicita uma confirmação (como: YES, NO, CANCEL)
showInputDialog	Solicita algum valor (possui um campo texto simples)
showMessageDialog	Informa ao usuário sobre algo (botão: OK)
showOptionDialog	Unificação dos três acima

A Classe JOptionPane

Estilos da Mensagem	Descrição
ERROR_MESSAGE	X num octógono
INFORMATION_MESSAGE	i num círculo
WARNING_MESSAGE	! num triângulo
QUESTION_MESSAGE	? num quadrado
PLAIN_MESSAGE	sem ícone

A Classe JOptionPane

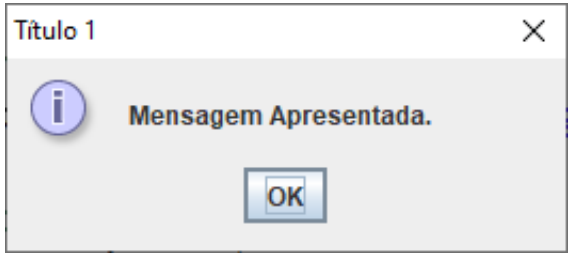
Estilos do Botão	Descrição
DEFAULT_OPTION	Ok
YES_NO_OPTION	Sim e Não
YES_NO_CANCEL_OPTION	Sim, Não e Cancelar
OK_CANCEL_OPTION	Ok e Cancelar

A Classe JOptionPane

Tipos de Respostas	Valor
YES_OPTION	0
NO_OPTION	1
CANCEL_OPTION	2
OK_OPTION	0
CLOSED_OPTION	-1

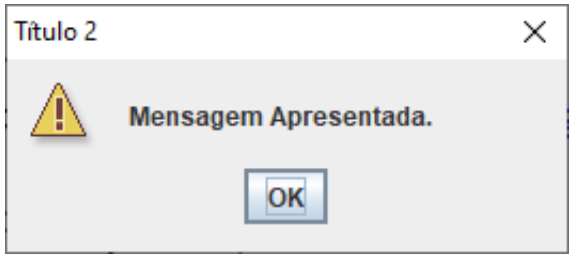
A Classe JOptionPane

Exemplos de JOptionPane:



Fonte: própria

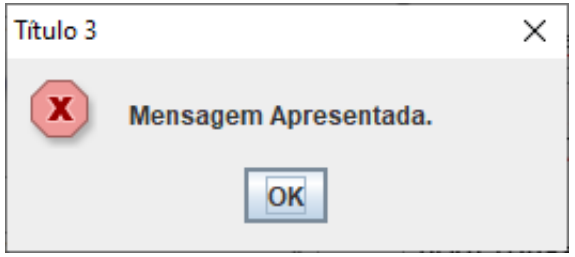
```
JOptionPane.showMessageDialog(null,
        "Mensagem Apresentada.",
        "Título 1",
        JOptionPane.INFORMATION_MESSAGE);
```



Fonte: própria

```
JOptionPane.showMessageDialog(null,
        "Mensagem Apresentada.",
        "Título 2",
        JOptionPane.WARNING_MESSAGE);
```

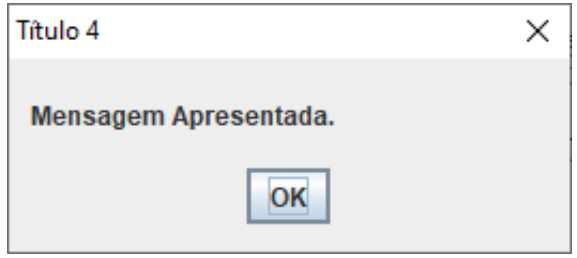
```
JOptionPane.showMessageDialog(null,
        "Mensagem Apresentada.",
        "Título 3",
        JOptionPane.ERROR_MESSAGE);
```



Fonte: própria

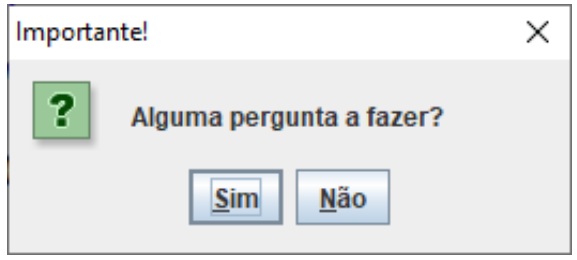
A Classe JOptionPane

Exemplos de JOptionPane:



Fonte: própria

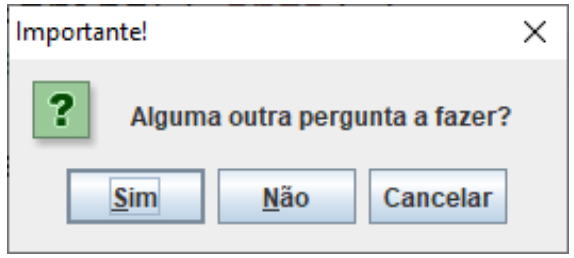
```
JOptionPane.showMessageDialog(null,
        "Mensagem Apresentada.",
        "Título 4",
        JOptionPane.PLAIN_MESSAGE);
```



Fonte: própria

```
int n = JOptionPane.showConfirmDialog(
        null,
        "Alguma pergunta a fazer?",
        "Importante!",
        JOptionPane.YES_NO_OPTION);
```

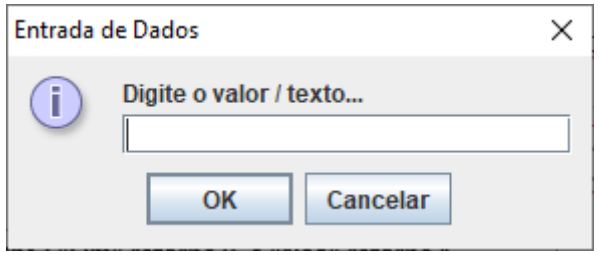
```
int n =
JOptionPane.showConfirmDialog(null,
        "Alguma outra pergunta a fazer?",
        "Importante!",
        JOptionPane.YES_NO_CANCEL_OPTION);
```



Fonte: própria

A Classe JOptionPane

Exemplos de JOptionPane:



Fonte: própria

```
String s = JOptionPane.showInputDialog(
        null,
        "Digite o valor / texto...",
        "Entrada de Dados",
        JOptionPane.INFORMATION_MESSAGE);
```

- Neste caso, como o retorno é uma String, retornará para “s” o texto digitado. Se nada for digitado, retornará vazio. Se clicar em “Cancelar” ou se fechar a janela de *dialog* (no “X”), então retornará para “s” o valor nulo (null).

Interatividade

Qual das opções abaixo caracteriza uma interface?

a) Uma interface é uma classe abstrata.
b) Uma interface somente possui métodos estáticos.
c) Existindo uma interface num sistema, todas as classes deste sistema deverão implementar seus métodos.
d) Uma interface não pode possuir atributos.
e) Os métodos existentes numa interface devem ser implementados nas classes que implementarem esta interface.

Resposta

Qual das opções abaixo caracteriza uma interface?

a) Uma interface é uma classe abstrata.
b) Uma interface somente possui métodos estáticos.
c) Existindo uma interface num sistema, todas as classes deste sistema deverão implementar seus métodos.
d) Uma interface não pode possuir atributos.
e) Os métodos existentes numa interface devem ser implementados nas classes que implementarem esta interface.

Wrapper Classes

- Wrapper vem do inglês “wrap” que significa envolver. Tem como principal função “envolver elementos” adicionando funcionalidades a eles.

- O Java possui diversos Wrappers que adicionam funcionalidades a outras classes ou tipos primitivos.

- Existem Wrappers: para tratar o fluxo de conexões (ObjectInputStream), para tratar fluxos de áudio (AudioInputStream), baseados em tipos primitivos (DataInputStream), ou para adicionar buffers (BufferedInputStream).

Wrapper Classes

- O foco aqui está para as representações de tipos primitivos.
- Com os Wrappers de tipos primitivos pode-se executar métodos como: parseTipo, valueOf entre outros.

Tabela que relaciona os tipos primitivos e suas respectivas "*wrapper classes*".
Fonte: própria

Tipo Primitivo	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Wrapper Classes

Transformando textos em números

- É possível transformar textos em número. Neste caso transforma-se a String em um tipo primitivo numérico a partir da sua Wrapper Class (desde que os valores das Strings representem um valor do tipo que se quer recuperar, pois do contrário gera um erro):

```
String s1, s2, s3;
s1 = "27";
int i = Integer.parseInt(s1);
long l = Long.parseLong(s1);
s2 = "27.98";
float f = Float.parseFloat(s2);
double d = Double.parseDouble(s2);
//int i = Integer.parseInt(s2);  // este geraria erro
s3 = "3956";
char c = s3.charAt(2); // pega o caracter ‘5’
int val = Character.getNumericValue(c);
```

Entrada de Dados

- Em Java, uma possível forma de entrar com valores durante a execução de um programa é utilizando-se o comando: showInputDialog(...)

```
variavel = OptionPane.showInputDialog (null, "Mensagem interna", "Título da Janela",
JOptionPane.PLAIN_MESSAGE);
```

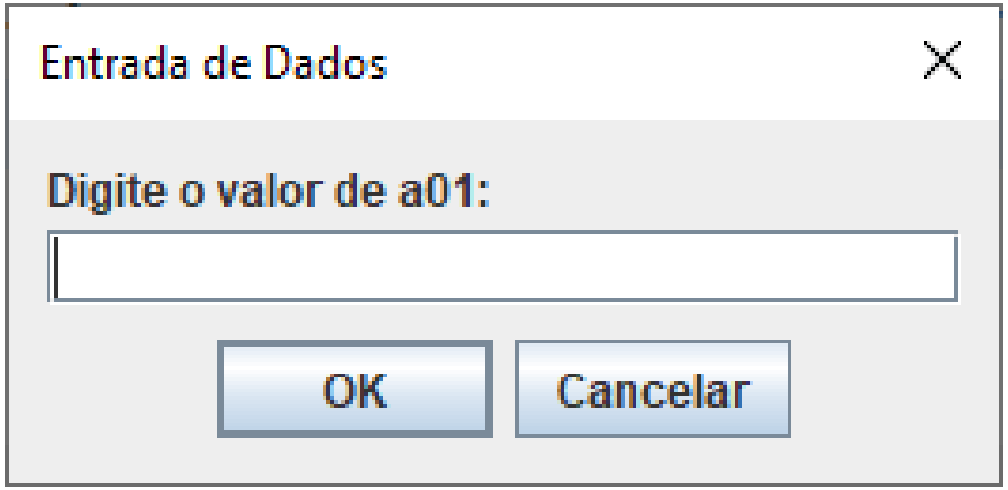
- A variável que recebe este valor deverá ser uma variável do tipo String.
  - O primeiro parâmetro (*null*) indica que este *Dialog* não está vinculado a nenhum *Frame* (item que será estudado em outro curso).
  - O segundo parâmetro determina a mensagem interna do *Dialog*.
  - O terceiro parâmetro determina o título da Janela de *Dialog*.
  - O quarto parâmetro determina se algum ícone específico deve ser mostrado (no caso, *PLAIN_MESSAGE* não mostra ícone algum).

Entrada de Dados

Exemplo:

```
String sa01 = JOptionPane.showInputDialog(null, "Digite o valor de a01:", "Entrada de Dados", JOptionPane.PLAIN_MESSAGE);
```

O exemplo acima gera a seguinte tela de *Dialog*:



Fonte: própria

Entrada de Dados

- Caso seja necessário se transformar o valor de entrada em número, pode-se utilizar as Classes representativas dos tipos primitivos (como por exemplo para os tipos int e double).

Um exemplo com um valor fixo:

```
String txtNum = "37";   //... Valor do tipo String
// ...transformando o valor em um número inteiro
int i01 = Integer.parseInt(txtNum);
// ...transformando o valor em um número real
double d01 = Double.parseDouble(txtNum);
```

- No entanto, se o valor inicial for impossível de ser transformado em número (como por exemplo: `txtNum = "3wk7"`), então a execução do programa resultaria numa Exceção do tipo “NumberFormatException”.

Interatividade

No programa abaixo, o que será impresso na tela da console se, quando ao ser executado, o usuário digitar o número 7 no campo de entrada de dados e em seguida clicar no botão OK?

```
String txt = JOptionPane.showInputDialog(null, "Digite um valor numérico:", "Entrada de Dados", JOptionPane.PLAIN_MESSAGE);
double d1 = Double.parseDouble(txt);
System.out.print(d1*2);
```

a) 7.
b) 7*2.
c) 14.0.
d) 7.0.
e) Na console aparecerá um erro (uma exceção) já que o valor digitado é uma String e não é possível se fazer contas com ele.

Resposta

No programa abaixo, o que será impresso na tela da console se, quando ao ser executado, o usuário digitar o número 7 no campo de entrada de dados e em seguida clicar no botão OK?

```
String txt = JOptionPane.showInputDialog(null, "Digite um valor numérico:", "Entrada de Dados", JOptionPane.PLAIN_MESSAGE);
double d1 = Double.parseDouble(txt);
System.out.print(d1*2);
```

a) 7.
b) 7*2.
c) 14.0.
d) 7.0.
e) Na console aparecerá um erro (uma exceção) já que o valor digitado é uma String e não é possível se fazer contas com ele.

ATÉ A PRÓXIMA!