



Desenvolvimento de Sistemas

Ambiente de programação: IDE, instalação e configuração

Introdução

Desde sua criação, no início da década de 1990, o Java é uma das linguagens mais populares do mundo e, ano após ano, continua mantendo sua posição entre as principais linguagens de programação do mundo graças à sua versatilidade e capacidade de se adaptar a quase todos os dispositivos e plataformas.

Todas essas características tornam os programadores Java muito valorizados no mercado de trabalho, que oferece facilidade para esses profissionais encontrarem colocação e salários acima da média de outras profissões que tenham o mesmo nível de formação.

Algumas vantagens da programação em Java são:

- ◆ Programação multiambiente (programe em Windows, Linux ou Mac)
- ◆ Programação multiplataforma (programe para *web*, *desktop*, *mobile*, *smartwatch*, TV etc.)
- ◆ Comunidade de usuários forte e colaborativa
- ◆ Grande número de *frameworks*
- ◆ Linguagens associadas (a JVM pode rodar outras linguagens além do Java, como Groovy, Scala, JPython e JRuby)
- ◆ Diversos IDEs (*integrated development environments*)

Neste conteúdo, você poderá rever algumas das características e ferramentas de programação Java, assim como verá recursos para o desenvolvimento de interfaces visuais e de interação.

Iniciando com Java

Para iniciar seus estudos, relembre o ecossistema Java e suas versões de acordo com o propósito destinado.

JSE (Java Standard Edition)

É a versão do Java que é usada para desenvolver aplicativos para *desktop*, para console ou que tenham interface gráfica.

JME (Java Micro Edition)

É a versão do Java para desenvolvimento de aplicações embarcadas, como roteadores, *switch* e outros.

Java TV

O Java TV é uma API (*application programming interface*) para desenvolvimento de aplicativos para *smart* TVs e TV Digital.

Javafx

Essa plataforma do Java foi desenvolvida para a criação de “aplicações ricas” para a Internet, ou seja, aplicações *web* que se comportam como aplicativos *desktop*. Os aplicativos gerados são poderosas aplicações multiplataforma.

JEE Java Enterprise Edition

Essa versão é um importante guarda-chuva de projetos de grande porte e para empresas. Nessa plataforma são encontrados o JSP (Java Server Pages) para executar o Java em aplicações *web*, o JPA (Java Persistence API) para armazenamento de informações e o JSF (Java Server Faces), que lida com a camada de visão em um projeto MVC (*model, views, controllers*).

Java Card

É uma plataforma para programação de *smart cards*, como os de bancos e outros.

É importante também lembrar os componentes presentes em uma plataforma Java:

JVM (Java Virtual Machine)

Responsável por rodar os programas feitos em Java, já que, diferentemente de outras linguagens, a linguagem JAVA não gera um binário executável, mas sim um *bit code*, um código intermediário que é interpretado pela máquina virtual.

JRE (Java Runtime Environment)

Ambiente de execução de aplicações Java, no qual estão incluídas, além da JVM, uma série de bibliotecas fundamentais para executar aplicações Java. É o mínimo a se instalar em um computador para se executar aplicação Java.

JDK (Java Development Kit)

Bibliotecas e ferramentas de desenvolvimento de Java, essenciais para o programador. Sem o JDK, você até pode escrever os códigos Java, mas não poderá testá-los ou distribuí-los como programas.

IDE

Toda linguagem de programação precisa de um compilador para transformar código-texto em um programa executável. Com Java, para programar, basta que você tenha JDK e use o compilador “javac”, presente no *kit*. No entanto, a tarefa de programar pode se tornar muito complexa para sistemas maiores caso não conte com ferramentas especializadas para isso.

É aí que se encaixam os IDEs.

IDE, que é uma sigla do inglês para *integrated development environment*, ou, em português, “ambiente de desenvolvimento integrado”, é um programa de computador que traz uma série de ferramentas de apoio para agilizar o desenvolvimento de *software*.

Existem vários IDEs para Java, mas aqui você seguirá usando o **NetBeans**, por ser um ambiente completo e já ter em seu instalador o JDK e o JRE integrados, facilitando a configuração do ambiente de trabalho.

Instalação e configuração

Reveja brevemente os passos para a instalação do IDE NetBeans para a sua máquina. Caso não tenha instalado, este é um bom momento para executar essa ação.

Iniciando um novo projeto Java *desktop*

Confira agora como começar um projeto Java *desktop*.

Abra o NetBeans e escolha a opção **File** e depois **New Project**.

Na janela que abre, selecione **Java with Ant**, em seguida **Java Application** e depois o botão **Next**.

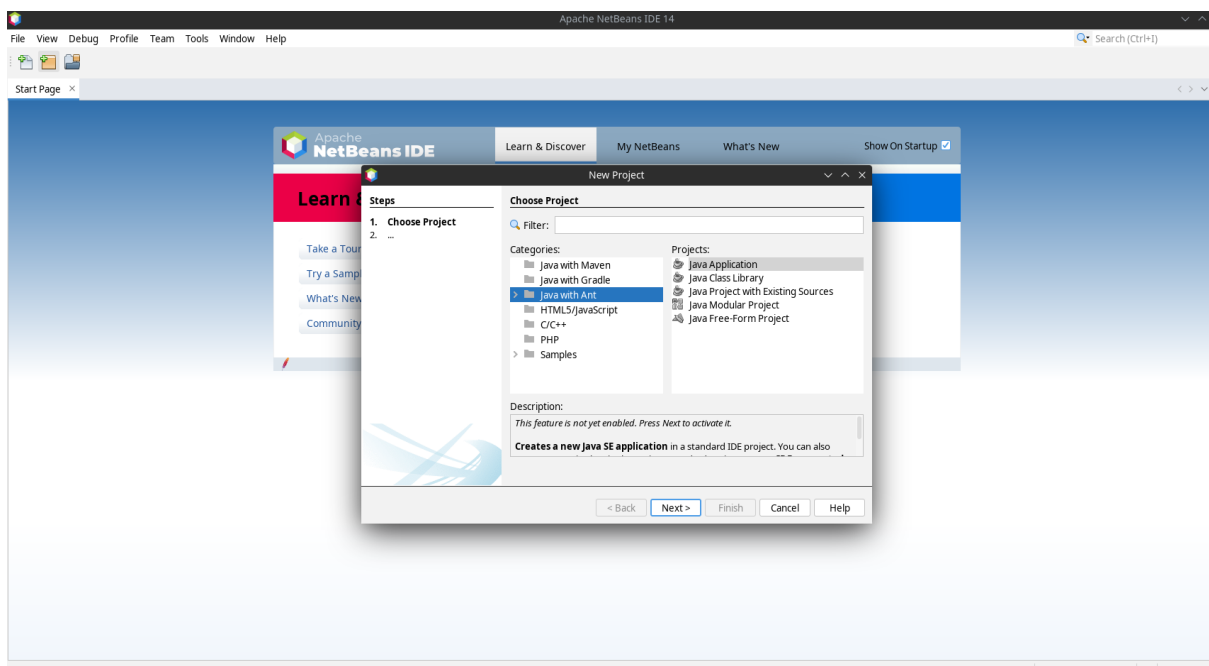


Figura 6 – Criando projeto no NetBeans

Fonte: NetBeans (2022)

Na próxima tela, dê um nome para o seu projeto, desmarque a opção **Create Main Class** e clique em **Finish**.

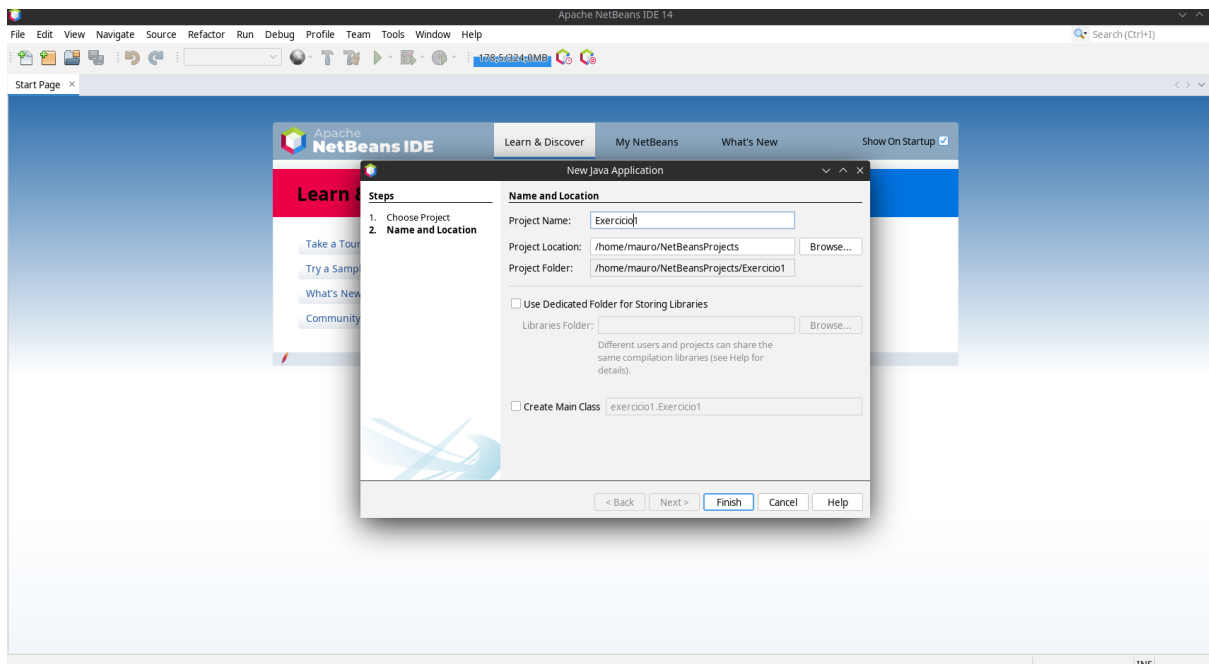


Figura 7 – Criando projeto no NetBeans

Fonte: NetBeans (2022)

Selecione o projeto no explorador de arquivos, clicando no nome do projeto.

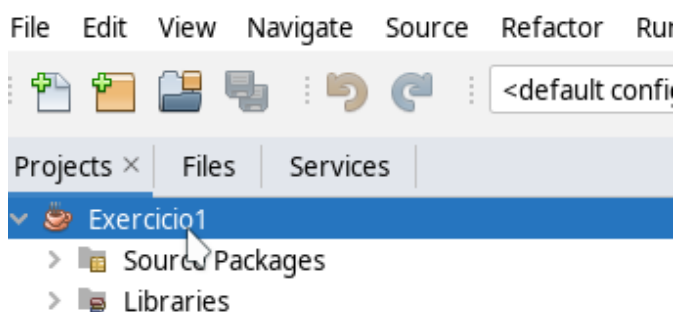


Figura 8 – Acessando o projeto criado

Fonte: NetBeans (2022)

Clique no botão **Novo Arquivo**, que é o primeiro na barra de botões.

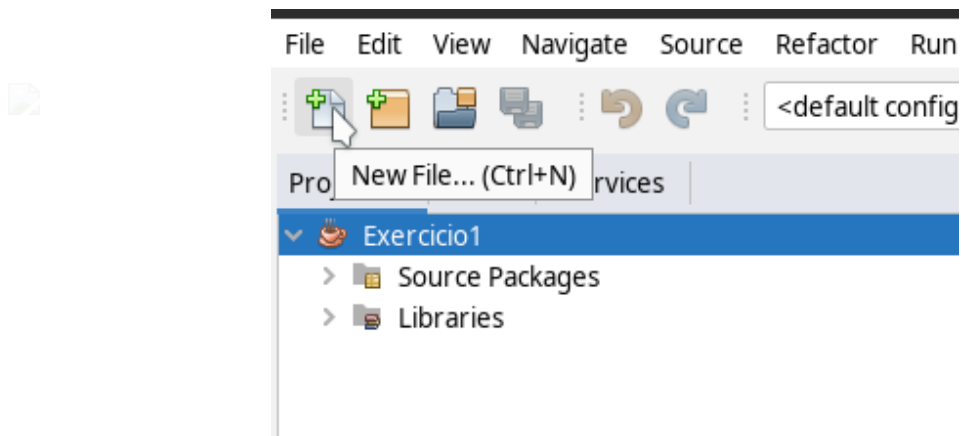


Figura 9 – Criando um novo arquivo

Fonte: NetBeans (2022)

Na janela que se abre, selecione **Java**, em seguida **Java Class** e clique em **Next**.

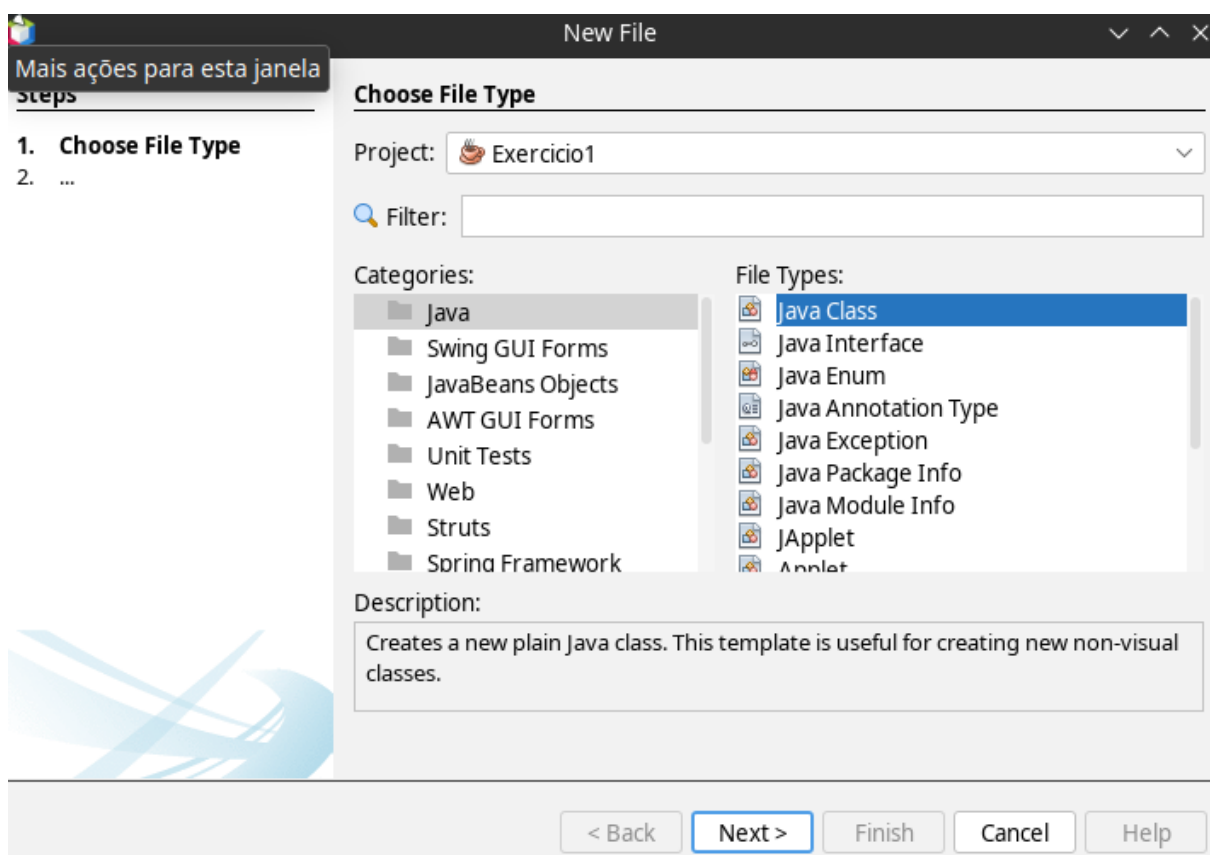


Figura 10 – Escolhendo tipo de arquivo

Fonte: NetBeans (2022)

Na tela seguinte, mude o nome da classe para **Main** e clique em **Finish**.

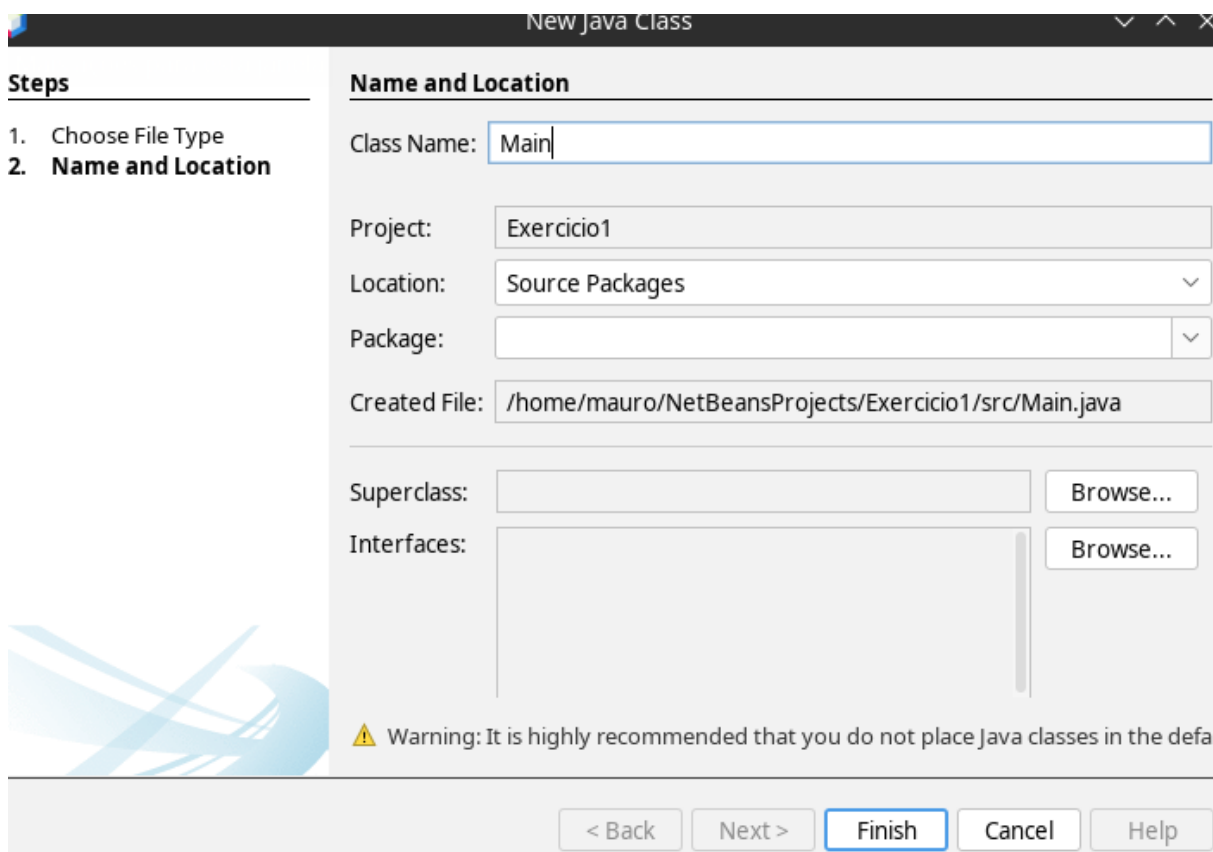


Figura 11 – Nomeando a classe

Fonte: NetBeans (2022)

Você deve ter uma tela semelhante a que está apresentada a seguir. Agora, será preciso concentrar-se na edição do código-fonte dentro da classe **Main** que foi criada.

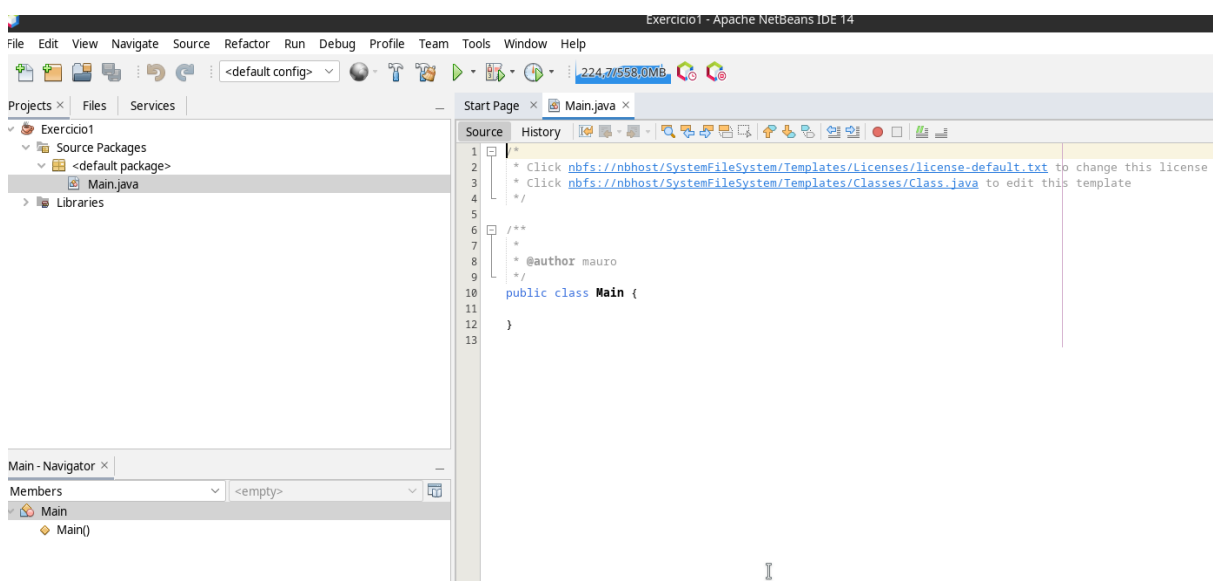


Figura 12 – Tela principal do NetBeans

Fonte: NetBeans (2022)



Iniciando com interface gráfica em Java

A linguagem Java é muito poderosa e contém diversos recursos para a criação de interfaces gráficas. Você verá um exemplo de como criar um pequeno aplicativo utilizando caixas de diálogo para exibição de mensagens e interação.

Como será utilizada a tecnologia Java Swing, será necessário importar o pacote no início do código, antes da abertura da classe.

Acrescente a seguinte linha:

```
import javax.swing.JOptionPane;
```

Depois, modifique o código existente para que fique da seguinte forma:

```
public class Main {  
    public static void main(String[] args)  
    {  
        JOptionPane.showMessageDialog(null, "Olá Mundo!!");  
    }  
}
```

Salve e clique em **Executar**. O resultado deve ser semelhante ao da tela a seguir:

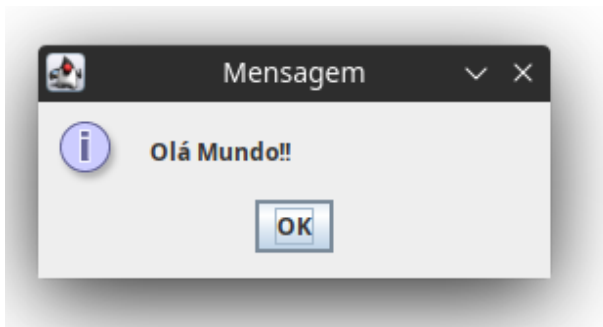


Figura 13 – Resultado da execução do código

Fonte: NetBeans (2022)

Isso não é interessante? Já foi criada a primeira tela, utilizando o `JOptionPane`, uma classe que oferece uma série de opções para montar janelas de diálogo semelhantes a essa. Algumas dessas opções disponibilizam interação com o usuário, com possibilidade de entrada de dados, por exemplo.

Criando um aplicativo com `JOptionPane`

Agora você estudará as mensagens do `JOptionPane` a partir do desenvolvimento de um pequeno aplicativo que **calculará o IMC do usuário e depois mostrará o tipo de dieta que ele deve fazer**.

Primeiro, é preciso criar uma tela de boas-vindas com *message dialog*, que serve apenas para mostrar mensagens sem interações. Para isso, será modificado o código anterior para que fique desta forma:

```
import javax.swing.JOptionPane;

public class Main {
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(null, "Olá\nVamos calcular a melhor dieta para você!!");
    }
}
```

Você notou o código `\n` no meio da mensagem? Esse conjunto de caracteres serve para criar uma quebra de linha no texto da mensagem.

Na sequência, serão solicitadas ao usuário informações para o cálculo. Neste caso, são necessários o nome, a altura e o peso do usuário. Essas informações serão armazenadas em variáveis, criadas logo acima da mensagem anterior.

```
import javax.swing.JOptionPane;

public class Main {
    public static void main(String[] args)
    {
        //Variáveis
        Double peso;
        Double altura;
        Double imc;
        String nome;
        String mensagem = "";

        //Janelas iniciais
        JOptionPane.showMessageDialog(null, "Olá\nVamos calcular a melhor dieta para você!!");
    }
}
```

Agora, será necessário montar os painéis que permitem que o usuário escreva as informações. Nesse caso, utilize a caixa de *input*, para acrescentar três *inputs* na sequência da primeira mensagem. Mas lembre-se de que as informações precisam ser armazenadas nas variáveis, então o código ficará um pouco diferente.

Outro ponto importante é que **tudo que o usuário escreve nos diálogos é *string***, então você deve converter o que ele escreveu para depois armazenar no tipo correto.

Aumente o código para que ele fique desta forma:

```
import javax.swing.JOptionPane;

public class Main {
    public static void main(String[] args)
    {
        //Variáveis
        Double peso;
        Double altura;
        Double imc;
        String nome;
        String mensagem = "";

        //Janelas iniciais
        JOptionPane.showMessageDialog(null, "Olá\nVamos calcular a melhor dieta para você!!");
        nome = JOptionPane.showInputDialog("Digite seu nome:");
        peso = Double.parseDouble(JOptionPane.showInputDialog("Informe seu peso em Kg:"));
        altura = Double.parseDouble(JOptionPane.showInputDialog("Informe sua altura em cm:"));

    }
}
```

Teste a aplicação para verificar se há algum erro de digitação e ver os diálogos de *input* que são semelhantes a esse.

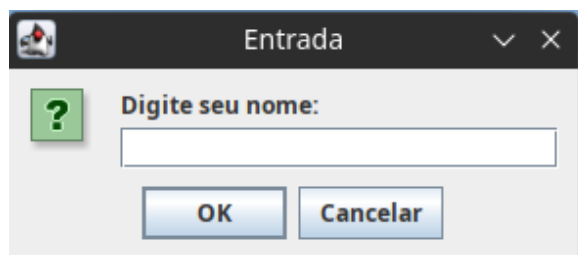


Figura 14 – Diálogo com entrada de dados

Fonte: NetBeans (2022)

Caso tudo esteja funcionando, você poderá processar os dados digitados pelo usuário. A lógica do cálculo do IMC é o peso dividido pela altura ao quadrado:

P/A²

Fazendo esse cálculo no Java, será preciso acrescentar a seguinte linha depois das linhas que formam as janelas:

```
imc = peso / (altura * altura);imc = peso / (altura * altura);
```

Ainda será necessário definir a mensagem que será mostrada para o usuário. Lembre-se de que, de acordo com o IMC da pessoa, a mensagem será diferente.

Neste passo, o código deve estar assim:

```
import javax.swing.JOptionPane;

public class Main {
    public static void main(String[] args)
    {
        //Variáveis
        Double peso;
        Double altura;
        Double imc;
        String nome;
        String mensagem = "";

        //Janelas iniciais
        JOptionPane.showMessageDialog(null, "Olá\nVamos calcular a melhor dieta para você!!");
        nome = JOptionPane.showInputDialog("Digite seu nome:");
        peso = Double.parseDouble(JOptionPane.showInputDialog("Informe seu peso em Kg:"));
        altura = Double.parseDouble(JOptionPane.showInputDialog("Informe sua altura em cm:"));

        // Cálculo
        imc = peso/(altura/100 * altura/100);
        if(imc < 18.5){
            mensagem = nome+" você está muito magro.\nPrecisa de uma dieta para engordar";
        }else if(imc < 24.9){
            mensagem = nome+" você está com peso ideal.\nNão precisa de dieta";
        }else if(imc < 29.9){
            mensagem = nome+" você está com sobrepeso.\nPrecisa de uma dieta para emagrecer";
        }else if(imc < 30){
            mensagem = nome+" você está com obesidade.\nPrecisa de uma dieta, exercícios e uma mudança de vida";
        }else {
            mensagem = nome+" você está com obesidade grave.\nPrecisa procurar um médico";
        }

        // Mensagem final
        JOptionPane.showMessageDialog(null, mensagem);

    }
}
```

Agora você já tem seu primeiro programa em Java para *desktop*!



Criando janelas via código

O Java conta com uma biblioteca chamada Swing, da qual faz parte inclusive a classe **JOptionPane** exercitada anteriormente. Essa biblioteca substitui uma mais antiga, destinada a interfaces gráficas, chamada AWT, e hoje se firma como padrão na linguagem para a construção de telas e interações.

A biblioteca baseia-se em componentes (classe-base Component) que podem ser contêineres (janelas, painéis, entre outros) ou elementos (caixa de texto, botões, rótulos e outros). Toda a interface visual programada em Java pode ser feita via código e esse é um bom exercício para entender os fundamentos presentes nessa programação.

Tudo deve se iniciar com uma janela, que em Java é definida pela classe **JFrame**. Um objeto **JFrame** definirá a janela visual da aplicação e permitirá a inclusão dos demais elementos que devem estar presentes na tela, como caixas de texto e botões. O objeto também definirá as características (tamanho, por exemplo) e o comportamento da janela (se pode ser redimensionada, por exemplo).

Para entender melhor, pratique com um exemplo.

Crie um projeto Java Ant chamado “JanelaJava”. Edite o método **main()** da classe principal, conforme a seguir:


```
import javax.swing.JFrame;

public class JanelaJava {

    public static void main(String[] args) {
        JFrame janela = new JFrame();
        janela.setSize(300, 200);
        janela.setVisible(true);
    }
}
```

- ◆ `JFrame janela = new JFrame();` – Cria em memória a janela, ou seja, o objeto de janela é criado.
- ◆ `janela.setSize(300, 200);` – Ajusta o tamanho da janela para largura de 300 pixels e altura de 200 pixels.
- ◆ `janela.setVisible(true);` – Torna a janela visível. Note que, se não houver esta linha de código, a janela não aparecerá na tela.

Ao rodar a aplicação, o resultado deve ser o seguinte:

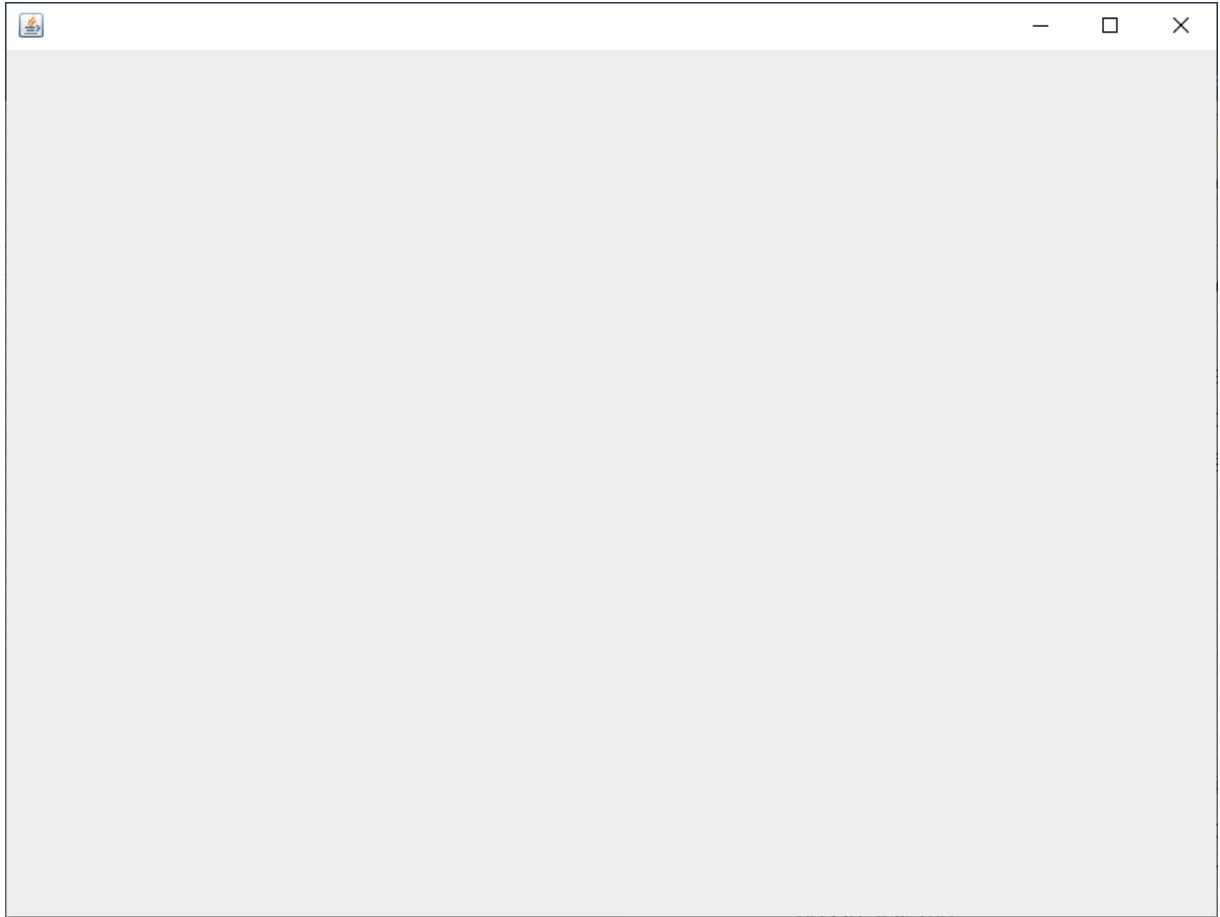


Figura 15 – Janela criada pelo código anterior

Fonte: NetBeans (2022)

Lembre-se de sempre incluir os *imports* necessários para o código escrito usando a ferramenta de sugestões do NetBeans.

Note que foi criada uma janela vazia, sem nem mesmo um título sobre ela. Teste ainda fechar a janela pelo botão **X** e observar no NetBeans, na aba **output**, que a aplicação continua rodando (você pode clicar no botão **Stop** na lateral esquerda da aba **output** para encerrar o programa). Isso acontece porque algumas propriedades da janela não foram definidas. Ajuste o código:

```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    janela.setVisible(true);  
}
```

Observe dois destaques no código anterior:

- ◆ O método **setTitle()** de **JFrame** (objeto “janela”) permite que se configure um título para aparecer na barra superior da tela.
- ◆ O método **setDefaultCloseOperation()** define o que o programa deve fazer quando se clicar no botão **X**. Por padrão, apenas esconderá a janela, mas o código está sendo configurado para que feche a aplicação (**EXIT_ON_CLOSE**).

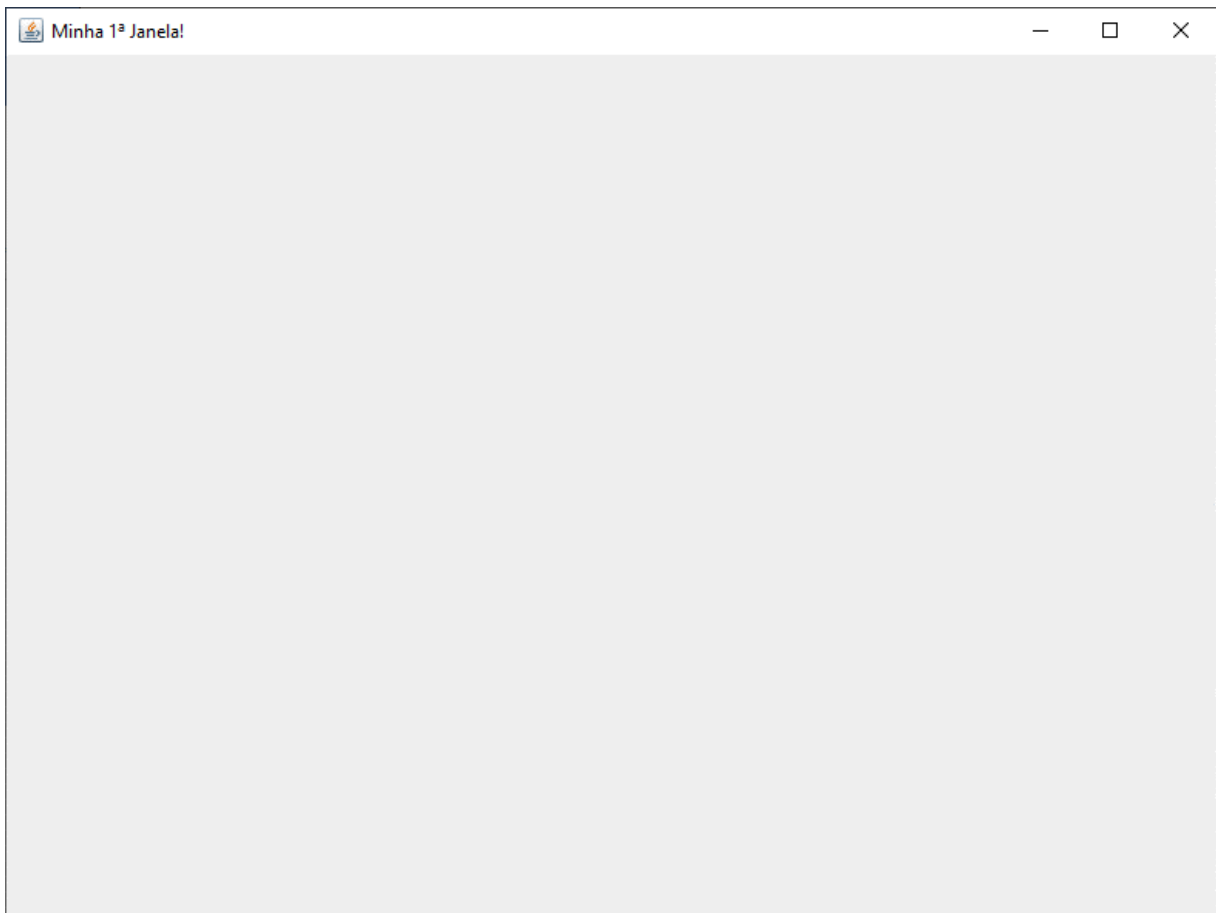


Figura 16 – Janela com título

Fonte: NetBeans (2022)

Se você testar a execução desse código, notará que a janela agora apresenta um título e o botão **X** encerra, de fato, o programa.

Como se pode então incluir elementos nesta janela? Tente embutindo um botão e uma etiqueta textual. Para o botão, utiliza-se a classe **JButton** da biblioteca Swing e, para rótulos, **JLabel**. Note a seguir as linhas destacadas no código da inclusão da etiqueta.

```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    //criando label  
    JLabel rotulo = new JLabel();  
    rotulo.setText("Dentro da Janela");  
  
    janela.setVisible(true);  
}
```

Rode a aplicação e note que não haverá nenhuma diferença na tela. Isso acontece porque, apesar de se estar criando a etiqueta, ela não está sendo associada à janela à qual ela deve pertencer. Para isso, será preciso usar o método **add()** da classe **JFrame**.

Veja a seguir como associar um componente à tela:

```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    //criando label  
    JLabel rotulo = new JLabel();  
    rotulo.setText("Dentro da Janela");  
    janela.add(rotulo);  
  
    janela.setVisible(true);  
}
```

Agora sim, o resultado será como nesta imagem:

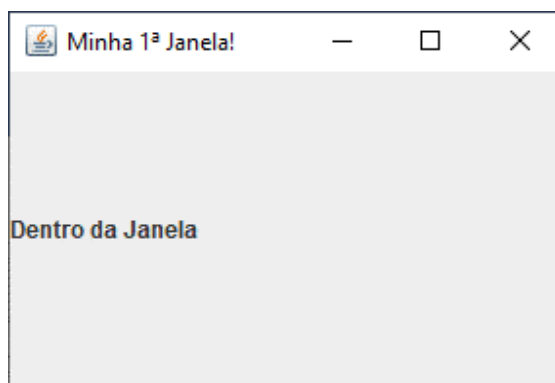


Figura 17 – Janela exibindo um rótulo “Dentro da Janela”

Fonte: NetBeans (2022)

Agora será incluído um botão na janela:

```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
    //criando label  
    JLabel rotulo = new JLabel();  
    rotulo.setText("Dentro da Janela");  
    janela.add(rotulo);  
  
    //criando botão  
    JButton botao = new JButton();  
    botao.setText("Clique aqui");  
    janela.add(botao);  
  
    janela.setVisible(true);  
}
```

O resultado não será exatamente como se poderia esperar. Note na imagem a seguir que o botão toma conta de toda a janela.

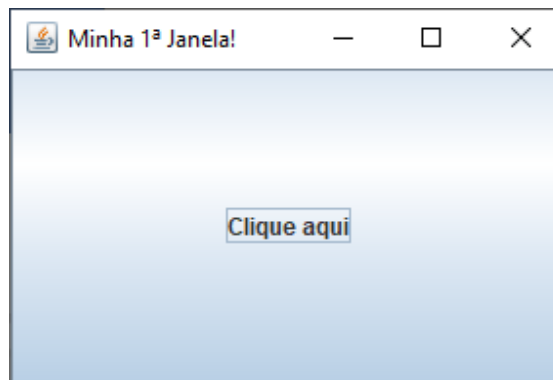


Figura 18 – Botão tomou conta da janela

Fonte: NetBeans (2022)

O fato aqui é que os elementos estão sendo posicionados e ajustados da maneira mais primitiva possível. É possível definir exatamente tamanhos e posições de cada elemento, mas também podem ser utilizados *layouts* do Java, que definem a ordem e

as posições em que os elementos são dispostos na tela. Um dos *layouts* mais básicos é o **FlowLayout**, que organiza os componentes visuais em linha e, caso não haja espaço na largura da janela, posiciona o componente uma linha abaixo. Pode-se configurá-lo como mostra o destaque no código a seguir:

```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    janela.setLayout(new FlowLayout());  
  
    //criando label  
    JLabel rotulo = new JLabel();  
    rotulo.setText("Dentro da Janela");  
    janela.add(rotulo);  
  
    //criando botão  
    JButton botao = new JButton();  
    botao.setText("Clique aqui");  
    janela.add(botao);  
  
    janela.setVisible(true);  
}
```

Quando se executa, o resultado será como o desta figura:

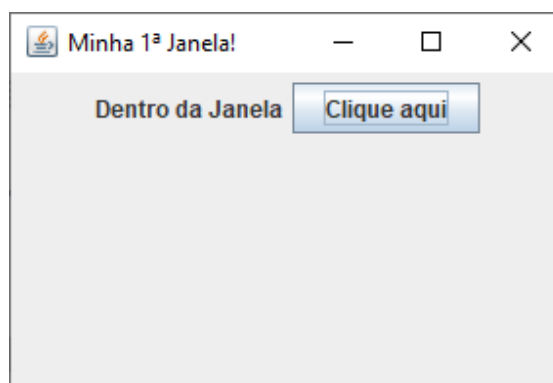



Figura 19 – Usando **FlowLayout**

Fonte: NetBeans (2022)

Existem vários *layouts* mais complexos e mais interessantes definidos na biblioteca de Java. Entre eles estão:

- 
- ◆ **BorderLayout** – Posiciona os componentes nas bordas (topo, base, esquerda, direita) e o restante de componentes ao centro.
 - ◆ **BoxLayout** – Posiciona os componentes em uma única linha ou coluna.
 - ◆ **GridLayout** – Posiciona os componentes como se estivessem em células de uma tabela, cujo número de linhas e colunas é definido.
 - ◆ **GridBagLayout** – Mais complexo que o **GridLayout**, permite mais flexibilidade entre células.

Para simplificar, neste momento sua atenção deverá estar voltada ao **FlowLayout**, porém, é importante que você use o exemplo anterior para testar alguns desses *layouts*, incluindo novos componentes na tela, se necessário.

Incluindo componente de entrada de dados

Para expandir esta pequena aplicação, inclua um componente de caixa de texto para capturar informação do usuário. Utilize para isso a classe **TextField**.


```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    janela.setLayout(new FlowLayout());  
  
    //criando label  
    JLabel rotulo = new JLabel();  
    rotulo.setText("Digite seu nome");  
    janela.add(rotulo);  
  
    //criando campo de entrada  
    JTextField campo = new JTextField(10);  
    janela.add(campo);  
  
    //criando botão  
    JButton botao = new JButton();  
    botao.setText("Clique aqui");  
    janela.add(botao);  
  
    janela.setVisible(true);  
}
```

Note no código que se trocou o texto da etiqueta e criou-se um objeto **JTextField** chamado “campo”, com dez colunas (uma medida de Java para definir a largura do campo).

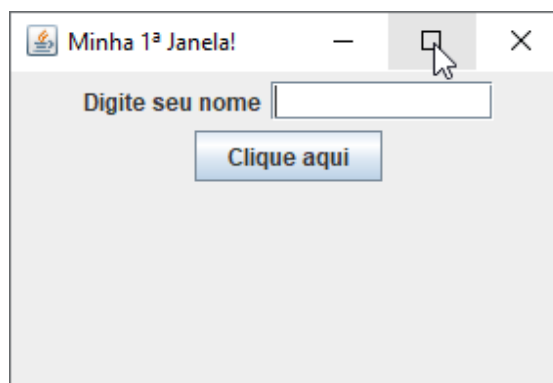


Figura 20 – Caixa de texto acrescentada à tela

Fonte: NetBeans (2022)

Em seguida, inclua ação no botão para usar, de alguma maneira, o valor digitado.

Implementando evento de botão via código



Até agora, sua aplicação não realiza nenhuma ação. Para exemplificar como se faz o botão executar uma operação, pode-se implementar algo muito simples: uma clássica mensagem de boas-vindas.

Implementam-se ações em botão usando o método **addActionListener()**. Você se lembra do padrão de projeto **Observer**? Esta é uma implementação próxima dele, na qual podem ser cadastrados objetos para que “ouçam” o componente e, quando uma ação acontecer, esses objetos executem um de seus métodos. No caso do botão, a ação esperada é o clique.

A sintaxe é a seguinte:

```
botao.addActionListener(objetoActionListener);
```

O objeto passado por parâmetro deve ser de uma classe que implemente a interface **ActionListener**. Assim, se você quiser incluir uma ação no botão, poderá criar uma nova classe para implementar essa ação. Experimente essa abordagem criando uma nova classe no pacote básico do projeto chamado **AcaoBotao**.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

class AcaoBotao implements ActionListener {

    private JTextField campoNome;

    public AcaoBotao(JTextField campoNome) {
        this.campoNome = campoNome;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "Boas vindas " + campoNome.getText()
+ "!");
    }
}
```

Como essa é uma classe separada e precisa-se do valor digitado, inclui-se uma referência ao campo (**JTextField**) no qual o nome será digitado e essa referência será preenchida no construtor da classe. No método **actionPerformed()**, que executará quando a ação (o clique) do botão ocorrer, obtém-se o valor digitado por meio do método **getText()** de **JTextField**.

No código de **main()**, então, aplica-se o método **addActionListener()**, como mostrado no destaque em azul do código a seguir:

```
public static void main(String[] args) {  
    JFrame janela = new JFrame();  
    janela.setSize(300, 200);  
    janela.setTitle("Minha 1ª Janela!");  
    janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    janela.setLayout(new FlowLayout());  
  
    //criando label  
    JLabel rotulo = new JLabel();  
    rotulo.setText("Digite seu nome");  
    janela.add(rotulo);  
  
    //criando campo de entrada  
    JTextField campo = new JTextField(10);  
    janela.add(campo);  
  
    //criando botão  
    JButton botao = new JButton();  
    botao.setText("Clique aqui");  
    janela.add(botao);  
  
    botao.addActionListener(new AcaoBotao(campo));  
  
    janela.setVisible(true);  
}
```

Em suma, você está fazendo um novo objeto **AcaoBotao** “escutar” o botão e executar seu método quando o botão clicar. Ao criar esse objeto, informa-se o campo de onde vem a informação que ele precisa. O resultado será semelhante ao que se vê na imagem a seguir:

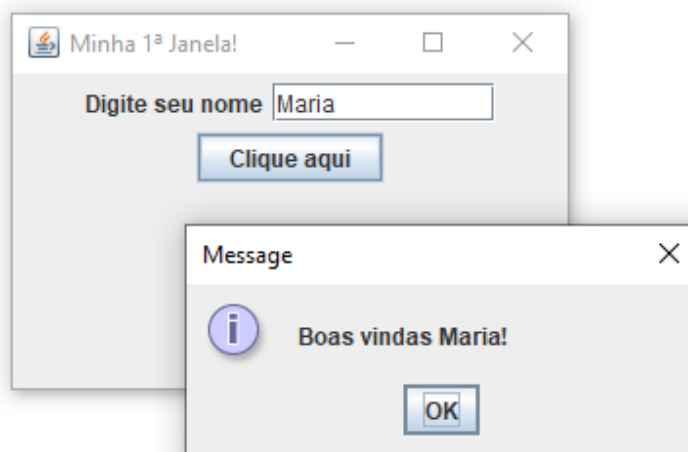


Figura 21 – O programa agora responde a uma ação do botão

Fonte: NetBeans (2022)

De fato, é possível implementar qualquer tipo de processamento a partir dessas bases. No entanto, há uma maneira um pouco mais direta de se implementar a ação em um botão, em que, ao invés de se criar toda uma classe separada, cria-se implicitamente uma implementação para a ação necessária. Use o trecho de código a seguir no lugar da chamada **botao.addActionListener()** que se tem até agora.

```
botao.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(janela, "Boas vindas " + campo.getText() +
"!");
    }
});
```

Está sendo aqui informado ao parâmetro de **AddActionListener** um novo objeto que, na verdade, implementa a interface **ActionListener** e seu método **actionPerformed()**. Note que se tem livre acesso aos componentes já criados na classe, como o **JTextField** “campo” e o **JFrame** “janela”.

Ao executar, o resultado será o mesmo mostrado na Figura 6, com a diferença de que a caixa de mensagem surgirá centralizada com relação à janela (isso se dá por conta do parâmetro “janela”, informado em **showMessageDialog()**). A classe **AcaoBotao**, criada anteriormente, neste momento pode até ser descartada.

Os componentes de Java Swing contam com uma variedade grande de eventos possíveis, como detecção de *mouse*, de foco ou de alteração de valor. À medida que você avançar em seus estudos, conhecerá alguns deles.

Criando interfaces gráficas com ferramentas de IDE

Você já aprendeu que pode criar programas usando caixas de mensagens. Também experimentou montar telas por meio de código, o que é uma alternativa muito melhor. No entanto, à medida que os componentes são incluídos, o código pode ficar mais complexo e o arranjo dos itens pode ser um bocado tedioso. Por isso, o NetBeans fornece uma ferramenta visual em que se pode clicar em componentes e arrastá-los, montando visualmente as telas do sistema.

Para isso, você iniciará mais um projeto **Java with Ant, Java Application**.

Dê o nome de “Exercicio2” para seu projeto e não se esqueça de desmarcar a opção **Criar Classe Principal**.

Que tal melhorar este projeto?

1. Acrescente um campo para o usuário inserir o nome dele e mostre na mensagem, junto com o IMC desse usuário, mais ou menos as informações a seguir. Utilize, é claro, a mensagem correta de acordo com o IMC do usuário:

"Olá Fulano, seu IMC é 26

Você está com peso ideal.

Não precisa de dieta”

2. Acrescente e faça funcionar um botão para limpar os campos e começar novamente.

Neste conteúdo, você aprendeu a configurar e a instalar o Java e o NetBeans em plataformas Windows e Linux. Além disso, aprendeu a criar tanto interfaces gráficas simples, com painéis **JOptionpane**, quanto interfaces gráficas com janelas, *frames* e

painéis. Você também utilizou botões, campos de inserção de textos e etiquetas, bem como os manipulou para receber dados do usuário e retornar respostas para ele com os dados processados.

A partir desses conhecimentos, você será capaz de produzir seus próprios programas de complexidade baixa e avançar para os próximos conhecimentos, nos quais encontrará a forma de desenvolver conteúdos de alta complexidade.

