Introdução a Classe, Métodos e Objetos

Para realizar uma tarefa em um programa é necessário um **método**. O **método** descreve os mecanismos que realmente realizam suas tarefas. Em Java, primeiro criamos uma unidade de programa chamada **classe** para abrigar um **método**. Em uma **classe**, você fornece um ou mais **métodos** que são projetados para realizar as tarefas da classe.

Você deve construir um **objeto** de uma **classe** antes de fazer um programa realizar as tarefas que a **classe** descreve como fazer.

Um **objeto** tem **atributos** que são carregados com o **objeto** quando ele é utilizado em um programa. Esses **atributos** são especificados como parte da **classe** de **objeto**. Os **atributos** são especificados pelas **variáveis** de **instância** da **classe**.

Vamos cria uma classe chamada Chamada que contém um método mensagem que exibe uma mensagem na tela.

```
// declaração da classe Chamada

public class Chamada
{
    // método que exibe uma mensagem de boas-vindas
    public void mensagem()
    {
        System.out.println("Bem vindo à Chamada");
    }// fim do método mensagem
```

```
} // fim da classe Chamada
```

A declaração do **método** começa com palavra-chave `public` para indicar que o método está “disponível para o público” - ele pode ser chamado a partir de métodos de outras classes. Em seguida, vem o **tipo de retorno** do método, que especifica o tipo de dados que o método retorna depois de realizar sua tarefa. Os parênteses depois do nome do método indicam que isso é um **método**. Os **parênteses vazios** indicam que esse método não requer informações adicionais para realizar sua tarefa.

Uma classe que contém o método `main` inicia a execução de um aplicativo Java. A classe `Chamada` não é um aplicativo, pois não contém `main`. Portanto, se tentar executar `Chamada` digitando `java Chamada` na janela de comando, você obterá uma mensagem de erro como essa:

```
adamsmith@MacBook-Pro-de-Adam Documents % javac Chamada.java
adamsmith@MacBook-Pro-de-Adam Documents % java Chamada
Erro: o método main não foi encontrado na classe Chamada; defina o método main como:
   public static void main(String[] args)
ou uma classe de aplicativo JavaFX deve expandir javafx.application.Application
adamsmith@MacBook-Pro-de-Adam Documents %
```

Ou, simplesmente:

`Exception in thread "main” java.lang.NoSuchMethodError: main`

Vamos declarar uma classe separada que contenha um método `main`.

```
// Criando um objeto Chamada e chamando seu metodo mensagem
public class ChamadaTest
{
    // método main inicia a execução
    public static void main(String[] args)
    {
        // criando um objeto de chamada e o atribui a minhaCh
        Chamada minhaChamada = new Chamada();
```

```
        // chama o método mensagem de minhaChamada
        minhaChamada.mensagem();
    } //fim do main
} //fim da classe ChamadaTest
```

Em geral, você não pode chamar um **método** que pertence a outra classe até criar um objeto dessa classe. Iniciamos com a **declaração da variável** `minhaChamada`. O t**ipo da variável** é `Chamada`.

A variável `minhaChamada` é inicializada com o resultado da expressão de criação de instância de classe `new Chamada()`. A palavra-chave `new` cria um objeto da classe especificada à direita da palavra-chave. Os parênteses à direita de `Chamada` são necessários. Eles em combinação com um nome de classe representam uma chamada para um **construtor**.

Você deve compilar as classes antes de executar o aplicativo. O comando pode ser:

`javac Chamada.java ChamadaTest.java`

ou pode complicar todas as classes no diretório

`javac *.java`

Declarando um método com um parâmetro

Um **método** pode exigir um ou mais **parâmetros** que representam informações adicionais necessárias para realizar a tarefa. Os **parâmetros** são definidos em uma lista de parâmetros separados por vírgula, que está localizado nos parênteses depois do nome do método. Todo parâmetro deve especificar um tipo e um identificador.

Iremos alterar o **método** `mensagem`:

```
// Declaração de classe com um método que tem um parâmetro
public class Chamada
{
```

```
    // exibe uma mensagem de boas-vindas para o usuário
    public void mensagem(String nomeCurso)
    {
        System.out.println("Bem vindo à Chamada do curso de "
    } // fim do método mensagem

} // fim da classe
```

```
//Cria objeto Chamada e passa uma String para seu método mens
import java.util.Scanner;

public class ChamadaTest
{
    // método main inicia a execução
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        Chamada minhaChamada = new Chamada();

        System.out.println("Coloque o nome do curso: ");
        String curso = input.nextLine();
        System.out.println(); // gera saída de uma linha em b

        // chama o método mensagem de minhaChamada e passa o
        minhaChamada.mensagem(curso);
    } // fim do método

} // fim da classe
```

O **método** `nextLine` lê os caracteres digitados pelo usuário até o caractere de nova linha ser encontrado, depois retorna um `String` que contém o caractere até, mas não incluindo, o caractere nova linha.

A **variável** `curso` entre parênteses é o argumento que é passado ao método `mensagem` para o método poder realizar sua tarefa. O valor da variável `curso` em

`main` torna-se o valor do parâmetro `nomeCurso` do **método** mensagem.

Há um relacionamento especial entre as **classes** que são compiladas no mesmo **diretório** no disco, como as classes `Chamada` e `ChamadaTest`. Por padrão, essas classes são consideradas pertencentes ao mesmo pacote - conhecido como o **pacote padrão**. As classes do mesmo pacote são **importadas** implicitamente para os arquivos de código-fonte de outras classes do mesmo pacote.

Variáveis de instância, métodos set e get

As **variáveis** declaradas no corpo de um método particular são conhecidas como **variáveis locais** e só podem ser utilizadas nesse método. Quando esse método terminar, os valores de suas variáveis locais são **perdidos**. Uma classe normalmente consiste em um ou mais métodos que manipulam os atributos que pertencem a um objeto particular da classe. Os atributos são representados como variáveis em uma declaração de classe. Essas variáveis são chamadas **campos** e estão declaradas dentro de uma declaração de classe, mas fora do corpo das declarações de método da classe. Quando cada objeto de uma classe mantém sua própria cópia de um atributo, o campo que representa o atributo também é conhecido como uma **variável de instância** — cada objeto (instância) da classe tem uma instância separada da variável na memória.

```
// classe Chamada que contém variável de instância

public class Chamada
{
    private String nomeCurso; // nome do curso para essa Cham

    //método para configurar o nome do curso
    public void setNomeCurso(String nome){
        nomeCurso = nome; // armazena o nome do curso
    } // fim do método set

    //método para recuperar o nome do curso
    public String getNomeCurso()
```

```
    {
        return nomeCurso;
    } // fim do método get

    //exibe uma mensagem de boas-vindas para o usuário
    public void mensagem()
    {
        //chama getNomeCurso para obter o nome do curso
        System.out.println("Bem vindo à Chamada do curso de "
    } //fim do método mensagem
} //fim da classe
```

Como a **variável** é declarada no corpo da classe, mas fora dos corpos dos métodos da classe, a declaração aqui é uma **variável de instância.**

A maioria das declarações de **variável de instância** é precedida pela palavra-chave `private`. As **variáveis** ou **métodos** declarados com o modificador de acesso `private` só são acessíveis a métodos da classe em que são declarados.

Observe que todas as instruções utilizam `nomeCurso`, embora ele não tenha sido declarado em nenhum dos métodos. Podemos utilizar o `nomeCurso` em métodos `Chamada` porque `nomeCurso` é uma **variável de instância** de classe. Observe também que a ordem em que os métodos são declarados em uma classe não determina quando eles são chamados em **tempo de execução**.

Note que um método de uma classe pode chamar outro método da mesma classe utilizando apenas o nome do método.

```
//Cria objeto Chamada e passa uma String para seu método mens
import java.util.Scanner;

public class ChamadaTest
{
    // método main inicia a execução
    public static void main(String[] args)
    {
```

```
        Scanner input = new Scanner(System.in);

        Chamada minhaChamada = new Chamada();

        // exibe valor inicial de nomeCurso
        System.out.println("Nome do curso inicial é " + minha

        // solicita e lê o nome do curso
        System.out.println("Coloque o nome do curso: ");
        String curso = input.nextLine();
        minhaChamada.setNomeCurso(curso); //configura o nome
        System.out.println(); // gera saída de uma linha em b

        // chama o método mensagem de minhaChamada e passa o
        minhaChamada.mensagem();
    } // fim do método

} // fim da classe
```

Inicialmente, exibe o valor do curso inicial que chama o **método** `getNomeCurso` do objeto. Observe que a primeira linha de saída mostra o nome `null`. Diferente das variáveis locais, que não são automaticamente inicializadas, todo campo tem um valor inicial padrão - um valor fornecido pelo Java quando você não especifica o valor inicial do campo. O valor padrão de um campo tipo `String` é **null**.

As classes costumam fornecer métodos `public` para permitir a clientes configurar (**set**) ou obter (**get**) **variáveis de instância** `private`.

Inicializando objetos com construtores

Cada classe que você declara pode fornecer um método especial chamado **construtor** que pode ser utilizado para inicializar um objeto de uma classe quando o objeto for criado. De fato, o Java requer uma chamada de **construtor**

para todo objeto que é criado. A palavra-chave `new` solicita memória do sistema para armazenar um objeto e então chama o **construtor da classe** correspondente para inicializar o objeto. A `Chamada` é indicada pelos **parênteses** depois do nome da classe. **Um construtor deve ter o mesmo nome que a classe.**

Por padrão, o compilador fornece um **construtor padrão** sem parâmetros em qualquer classe que não inclui explicitamente um construtor. Quando uma classe tem somente o construtor padrão, suas variáveis de instância são inicializadas de acordo com seus valores padrões.

```
// classe Chamada que contém variável de instância

public class Chamada
{
    private String nomeCurso; // nome do curso para essa Cham

    // o construtor inicializa nomeCurso
    public Chamada(String nome)
    {
        nomeCurso = nome; // inicaliza o nome do curso
    } // fim do construtor

    //método para configurar o nome do curso
    public void setNomeCurso(String nome){
        nomeCurso = nome; // armazena o nome do curso
    } // fim do método set

    //método para recuperar o nome do curso
    public String getNomeCurso()
    {
        return nomeCurso;
    } // fim do método get

    //exibe uma mensagem de boas-vindas para o usuário
    public void mensagem()
    {
```

```
        //chama getNomeCurso para obter o nome do curso
        System.out.println("Bem vindo à Chamada do curso de "
    } //fim do método mensagem
} //fim da classe
```

```
//Cria objeto Chamada e passa uma String para seu método mens
import java.util.Scanner;

public class ChamadaTest
{
    // método main inicia a execução
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);

        // cria objeto Chamada
        Chamada minhaChamada = new Chamada("Física");

        // exibe valor inicial de nomeCurso
        System.out.println("Nome do curso inicial é " + minha

        // solicita e lê o nome do curso
        System.out.println("Coloque o nome do curso: ");
        String curso = input.nextLine();
        minhaChamada.setNomeCurso(curso); //configura o nome
        System.out.println(); // gera saída de uma linha em b

        // chama o método mensagem de minhaChamada e passa o
        minhaChamada.mensagem();
    } // fim do método

} // fim da classe
```

Uma **diferença** importante entre **construtores** e **métodos** é que os construtores **não podem retornar valores**, portanto, não podem especificar um

tipo de retorno (nem mesmo `void`). Normalmente, os construtores são declarados `public`. Se uma classe não incluir um construtor, as **variáveis de instância** da classe são inicializadas como seus **valores padrão**.

Se você declarar qualquer construtor para uma classe, o compilador Java não criará um construtor padrão para essa classe.

Exibindo texto numa caixa de diálogo

Os programas apresentados até agora exibem a saída na **janela de comando**. Muitos aplicativos utilizam janelas ou **caixas de diálogo** (também chamadas **diálogos**) para exibir a saída. Tipicamente, as caixas de diálogo são janelas nas quais os programas exibem mensagens importantes aos usuários. A classe `JOptionPane` fornece caixas de diálogo pré-construídas que permitem aos programas exibir janelas que contêm mensagens — essas janelas são chamadas de **diálogos de mensagem**.

```
// Imprimindo múltiplas linhas na caixa de diálogo
import javax.swing.JOptionPane;

public class Dialogo
{
    public static void main(String[] args)
    {
        //pede para o usuário inserir o nome
        String nome = JOptionPane.showInputDialog("Qual seu n

        // exibe a mensagem para cumprimentar o usuário
        JOptionPane.showMessageDialog(null, "Bem Vindo ao Jav
    } // fim do main
} // fim da classe
```

O programa utiliza a classe `JOptionPane` do pacote `javax.swing`. Esse pacote contém muitas classes que ajudam-lhe a criar **interfaces gráficas com o usuário** (Graphical User Interfaces — GUIs) para aplicativos. **Componentes GUI** facilitam a entrada de dados pelo usuário de um programa e apresentação

das saídas ao usuário. O método `JOptionPane.` **`showMessageDialog`** para exibir uma caixa de diálogo que contém uma mensagem. O método requer dois argumentos. O primeiro ajuda o aplicativo Java a determinar onde posicionar a caixa de diálogo. Se o primeiro argumento for `null`, a caixa de diálogo será exibida no **centro da tela**. O segundo argumento é a `String` a ser exibida na caixa de diálogo.

O método `JOptionPane.showMessageDialog` é o chamado de **método** **`static`**. Esses métodos muitas vezes definem tarefas frequentemente utilizadas. Note que você não cria um objeto da classe `JOptionPane` para utilizar seu método `static` `showMessageDialog`.

O aplicativo usa o método **`showInputDialog`** de `JOptionPane` para exibir uma caixa de diálogo de entrada que contém um **prompt** e um campo (conhecido como **campo de texto**) no qual o usuário pode inserir o texto. O argumento do método `showInputDialog` é o **prompt** que indica o nome que o usuário deve inserir. O usuário digita caracteres no campo de texto, depois clica no botão **OK** ou pressiona a tecla **Enter** para retornar a `String` para o programa. O método `showInputDialog` retorna uma `String` contendo os caracteres digitados pelo usuário.

Nota: se você pressionar o botão do diálogo **Cancel** ou pressionar a tecla *Esc*, o método retornará `null` e o programa exibirá a palavra **"null"** como nome.

Exercícios da Aula

1. Crie um aplicativo que mantém o saldo de uma conta bancária, realiza depósito e mostra o saldo da conta. Realize depósitos e apresente ao usuário seu saldo. (Dica: utilize o método `nextDouble` da classe `Scanner`.)
2. Refaça os exercícios da seção anterior que utiliza a classe Scanner com a classe `JOptionPane`. (Dica: utilize o método `static` `parseInt` da classe `Integer` para converter `String` para `Int`).

3. Modifique a classe Chamada com: Inclua uma variável de instância `String` que representa o nome do professor do curso; Forneça os métodos **set** e **get** para alterar e recuperar o nome do professor, respectivamente; Modifique o construtor para especificar dois parâmetros; Modifique o método chamada para gerar a saída de mensagem para o nome do curso e nome do professor.
4. Crie uma classe chamada `Date` que inclua três variáveis de instância — mês (tipo `int`), dia (tipo `int`) e ano (tipo `int`). Forneça um construtor que inicializa as três variáveis de instância e suponha que os valores fornecidos estejam corretos. Forneça um método ***set*** e um ***get*** para cada variável de instância. Forneça um método `displayDate` que exibe o mês, o dia e o ano separados por barras normais (/). Escreva um aplicativo de teste chamado `DateTest` que demonstra as capacidades da classe `Date`.