Classes e Métodos – Parte 1
a.k.a.: Organizando a Bagunça

Prof. Bruno Nogueira

Vamos começar a pensar orientado a objetos

- Nossos programas se resumiam ao método principal
  - Não explorávamos o potencial de programação orientada a objetos
  - Tudo em uma única classe
  - Classes eram "blocos de código", apenas
  - Esquema parecido com programação procedimental

- Não precisamos de muito esforço para perceber que essa não é a melhor maneira de se criar um programa em Java
  - Tudo em um único método fica confuso e poluído
  - Chegamos e escrever um único método com mais de 50, 60 linhas!

Classes e Métodos

- Vamos partir da seguinte analogia: você guiando um carro. Você quer fazê-lo andar mais rápido. O que faz? Pisa no acelerador
  - Antes disso acontecer, alguém teve que **projetar** este carro
    - Desenhos de engenharia, semelhantes às plantas de casa
    - Nesses desenhos de engenharia, inclui-se o pedal de acelerador, para acelerar o carro
  - Pedal "**oculta**" do motorista os mecanismos complexos que fazem o carro ir mais rápido
    - Isso permite que pessoas com pouco ou nenhum conhecimento de mecânica consigam guiar os carros
  - O carro, entretanto, não se acelera sozinho. Precisa que o motorista pise no pedal do acelerador

Classes e Métodos: Programação Orientada a Objetos

- Para realizar uma tarefa em um programa, é necessário um **método**
  - Método descreve os mecanismos que realizam a sua tarefa
  - Oculta do usuário as tarefas complexas que ele realiza
- Criamos uma unidade de programa chamada **classe** para abrigar os métodos
  - Assim como os desenhos de engenharia do carro abrigam o projeto do pedal do acelerador
  - Em uma classe, um ou mais métodos são fornecidos para realizar as tarefas daquela classe
  - Por exemplo, em uma classe que representa uma conta bancária, pode ter um método para depósito, um para saque e um para saber o saldo atual

Classes e Métodos: Programação Orientada a Objetos

- Assim como você não pode dirigir um projeto de engenharia, você não pode "dirigir" uma classe
  - Alguém tem que construir um caro a partir do projeto de engenharia, bem como você tem que construir um **objeto** de uma classe para realizar as tarefas que a classe descreve
- Ao dirigir o carro, o ato de pressionar o acelerador envia uma mensagem para o carro realizar uma tarefa – andar mais rápido
  - Em POO, de maneira análoga, você envia **mensagens** para um objeto por meio de **chamadas de método**
    - Instru um método do objeto a realizar sua tarefa

Atributos

- Um carro, além de suas capacidades, também tem muitos **atributos** que o descrevem
  - Cor, número de portas, quantidade de gasolina no tanque, velocidade atual (velocímetro), total de quilômetros percorridos (hodômetro), etc.
  - Estes atributos também são representados como parte do diagrama de engenharia
  - Cada carro mantém seus próprios atributos. Um carro sabe, por exemplo, a quantidade de gasolina que há no seu tanque, mas não sabe a dos outros carros

Atributos: Programação Orientada a Objetos

- Em POO, analogamente, tem-se atributos que são carregados com o objeto quando ele é utilizado
  - São especificados como parte da classe do objeto
    - Por exemplo: objeto "conta bancária" tem o atributo saldo, que representa a quantidade de dinheiro na conta
    - Cada objeto "conta bancária" sabe o saldo da conta que ele representa, mas não sabe os saldos de outras contas no banco
  - Atributos são especificados pelas **variáveis de instância** da classe

Métodos

```
1  public class ExibeMensagem
2  {
3      public void exibeMensagem()
4      {
5          System.out.println("Seja bem vindo, usuário!");
6      }
7  }
```

- A palavra chave public é um **modificador de acesso** – veremos mais sobre isso posteriormente
  - Indica que está "disponível para o público" – pode ser chamado a partir de métodos de outras classes

Métodos

```
1  public class ExibeMensagem
2  {
3      public void exibeMensagem()
4      {
5          System.out.println("Seja bem vindo, usuário!");
6      }
7  }
```

- Em seguida, vem o tipo de retorno do método
  - Especifica o tipo de dados que o método retorna ao seu método chamador depois de realizar a sua tarefa
  - O tipo de retorno void indica que esse método realizará uma tarefa que não retornará nenhuma informação para o seu método chamador
    - Métodos nextInt() e nextDouble() da classe Scanner retornam, respectivamente, um valor int e um double ao método chamador

Métodos

```
1  public class ExibeMensagem
2  {
3      public void exibeMensagem()
4      {
5          System.out.println("Seja bem vindo, usuário!");
6      }
7  }
```

- Nome do método
  - Geralmente, iniciam com letra minúscula. Palavras subsequentes começam com letras maiúsculas
- Parênteses
  - Indicam que é um método
  - Parênteses vazios indicam que o método não necessita de nenhuma informação adicional (parâmetro) para realizar a sua informação

Executando o método

```
1  public class ExibeMensagem
2  {
3      public void exibeMensagem()
4      {
5          System.out.println("Seja bem vindo, usuário!");
6      }
7  }
```

- Execução de um aplicativo começa pelo método main
  - Nesse exemplo, a classe ExibeMensagem não contém o método main. Logo, não podemos executar essa classe
  - Quando tentamos executar essa classe, temos o seguinte erro: java.lang.NoSuchMethodError: main
  - Devemos declarar uma outra classe, com um método main

Instanciando objetos

```
1   public class ExibeMensagemTeste
2   {
3       public static void main (String [] args)
4       {
5           // cria um objeto da classe ExibeMensagem e o atribui a exibeMsg
6           ExibeMensagem exibeMsg = new ExibeMensagem();
7
8           // chama o método exibeMensagem de exibeMsg
9           exibeMsg.exibeMensagem();
10      }
11  }
```

- Variável *exibeMsg* é do tipo classe *ExibeMensagem*
  - `exibeMsg` é inicializada com o resultado da **expressão de criação de instância de classe** `new ExibeMensagem()`
  - **new** cria um novo objeto da classe especificada à direita

Instanciando objetos

```
1  public class ExibeMensagemTeste
2  {
3      public static void main (String [] args)
4      {
5          // cria um objeto da classe ExibeMensagem e o atribui a exibeMsg
6          ExibeMensagem exibeMsg = new ExibeMensagem();
7
8          // chama o método exibeMensagem de exibeMsg
9          exibeMsg.exibeMensagem();
10     }
11 }
```

- Combinação do nome de classe com parênteses representam uma chamada ao **construtor** da classe
  - Método especial, que só é chamado na instanciação (inicialização dos dados) de um objeto

Utilizando objetos

```
1   public class ExibeMensagemTeste
2   {
3       public static void main (String [] args)
4       {
5           // cria um objeto da classe ExibeMensagem e o atribui a exibeMsg
6           ExibeMensagem exibeMsg = new ExibeMensagem();
7
8           // chama o método exibeMensagem de exibeMsg
9           exibeMsg.exibeMensagem();
10      }
11  }
```

- Podemos chamar métodos de objetos utilizando o **ponto separador** (.) após o nome do objeto
  - Já utilizamos e outras oportunidades (teclado.nextInt(), System.out.println(), etc)

Compilando mais de um arquivo

- Deve-se compilar todos os arquivos, separadamente
  - Deve-se compilar por ordem inversa de dependência (primeiro as classes que não dependem de nenhuma outra; em seguida, classes que utilizam objetos das outras classes)

```
javac ExibeMensagem.java ExibeMensagemTeste.java
```

  - Alternativamente, pode-se compilar todos os arquivos .java que estão na pasta corrente

```
javac *.java
```

Diagrama de classes UML

- Podemos representar nossas classes utilizando diagramas de classes UML (Unified Modeling Language)
  - Linguagem gráfica utilizada por programadores para representar sistemas orientados a objetos de maneira padronizada
  - Três compartimentos:

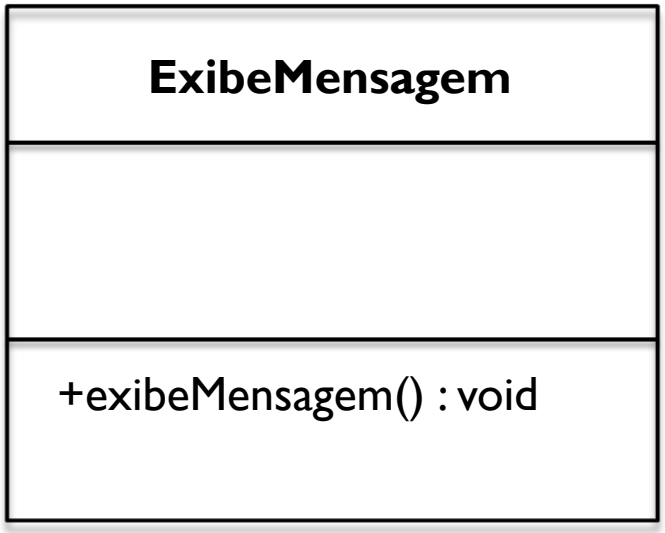
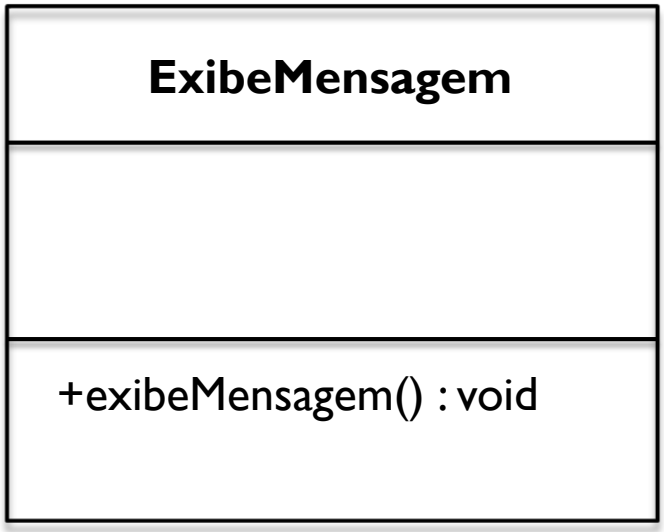


Diagrama de classes UML

- Neste exemplo:
  - Nome da classe é ExibeMensagem
  - Não há atributos
  - Método exibeMensagem
    - Modificador de acesso é público, por isso o símbolo (+) antes do nome do método



Métodos com parâmetros

- Voltando à nossa analogia: você quer fazer um carro acelerar e pra isso já sabe que deve pressionar o acelerador
- A questão é: quanto o carro deve acelerar?
  - Quanto mais pressionar o pedal, mais ele irá acelerar
- Mensagem para o carro acelerar deve incluir informações adicionais indicando a quantidade de aceleração desejada
- Essas informações adicionais são conhecidas como **parâmetros**

Métodos com parâmetros

- Parâmetros são definidos em uma lista de parâmetros, separados por vírgula, que está localizada nos parênteses depois do nome do método
  - É possível receber de 0 a infinitos parâmetros
  - Parênteses vazios, como visto anteriormente, indicam 0 parâmetros
  - Exemplo de método com parâmetro: Método `System.out.println` exige um argumento que especifica quais são os dados a serem exibidos

Métodos com parâmetros

- Modificações das classes anteriores para receber parâmetros

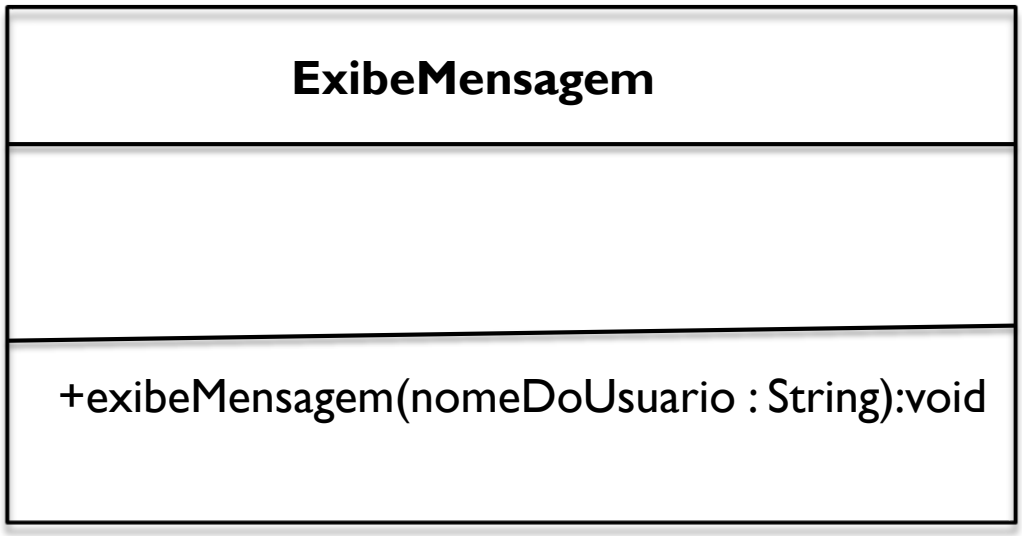
```java
public class ExibeMensagemComParametro
{
    public void exibeMensagem(String nomeDoUsuario)
    {
        System.out.printf("Seja bem vindo, %s!", nomeDoUsuario);
    }
}
```

```java
public class ExibeMensagemComParametroTeste
{
    public static void main (String [] args)
    {
        // cria um objeto da classe ExibeMensagemComParametro e o atribui a exibeMsg
        ExibeMensagemComParametro exibeMsg = new ExibeMensagemComParametro();

        // chama o método exibeMensagem de exibeMsg
        exibeMsg.exibeMensagem("Bruno");
    }
}
```

Métodos com parâmetros

- Parâmetros são especificados no diagrama UML



```java
public class Cachorro
{
    public String nome;
    public String raca;
    public int idade;

    // escreve os valores dos atributos na tela
    // não retorna nenhum valor - método void
    public void escreveSaida()
    {
        System.out.println("Nome: " + nome);
        System.out.println("Raça: " + raca);
        System.out.println("Idade: " + idade);
        System.out.println("Idade em anos humanos: " + getIdadeHumana());
    }

    // calcula o valor da idade de um cachorro em anos humanos
    // retorna valor inteiro correspondente à idade
    public int getIdadeHumana()
    {
        int idadeHumana = 0;
        if(idade <= 2)
            idadeHumana = idade * 11;
        else
            idadeHumana = 22 + ((idade - 2) * 5);
        return idadeHumana;
    }
}
```

Exemplo

```
1   public class CachorroTeste
2   {
3       public static void main (String [] args)
4       {
5           Cachorro barney = new Cachorro();
6           barney.nome = "Barney";
7           barney.raca = "Shih Tzu";
8           barney.idade = 2;
9           barney.escreveSaida();
10
11          int idadeHumana = barney.getIdadeHumana();
12          System.out.printf("A idade humana de %s é %d.", barney.nome, idadeHumana);
13      }
14  }
```

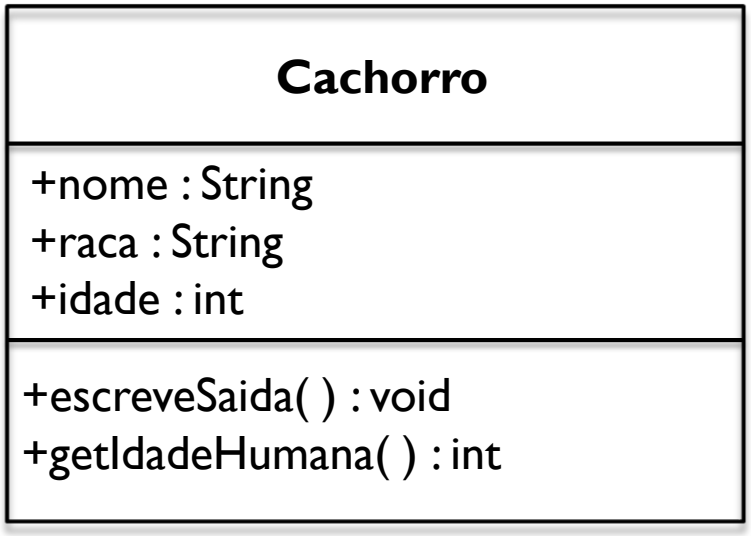
Exemplo

- Métodos que retornam valor devem especificar o tipo retornado e conter uma diretiva **`return`** seguida de um valor do tipo a ser retornado

```
public int getIdadeHumana()
{
    int idadeHumana = 0;
    if(idade <= 2)
        idadeHumana = idade * 11;
    else
        idadeHumana = 22 + ((idade - 2) * 5);
    return idadeHumana;
}
```

Exemplo

- UML da classe Cachorro



Chamando métodos

- Se um método retorna um valor, você pode chamá-lo em qualquer lugar em que pode usar o tipo de valor retornado pelo método

```
int idadeHumana = barney.getIdadeHumana();
```

- Métodos void, por outro lado, são invocados e seguidos por um ponto e vírgula (;)

```
barney.escreveSaida();
```

Variáveis de instância

- Variáveis que usamos nos nossos programas anteriores a essa aula eram declarados dentro do método main
  - Variáveis declaradas no corpo de um método são **variáveis locais** e só podem ser utilizadas neste método
  - Quando o método terminar, os valores das suas variáveis são perdidos
- **Variáveis de instância** (**atributos**) da classe são declarados no começo da mesma e têm validade ao longo de toda a classe
  - Fora da declaração de métodos
  - Cada objeto mantém sua própria cópia de um atributo

```java
public class Cachorro
{
    public String nome;
    public String raca;
    public int idade;

    // escreve os valores dos atributos na tela
    // não retorna nenhum valor - método void
    public void escreveSaida()
    {
        System.out.println("Nome: " + nome);
        System.out.println("Raça: " + raca);
        System.out.println("Idade: " + idade);
        System.out.println("Idade em anos humanos: " + getIdadeHumana());
    }

    // calcula o valor da idade de um cachorro em anos humanos
    // retorna valor inteiro correspondente à idade
    public int getIdadeHumana()
    {
        int idadeHumana = 0;
        if(idade <= 2)
            idadeHumana = idade * 11;
        else
            idadeHumana = 22 + ((idade - 2) * 5);
        return idadeHumana;
    }
}
```

Variáveis de instância

Variável local

Exercício

- Faça uma classe em Java que represente o seguinte diagrama de classes. Os métodos getNome, getRga e getCurso devem retornar os valores das variáveis. O método imprimeResumo deve imprimir na tela o resumo das informações do aluno.