LINGUAGEM DE PROGRAMAÇÃO I

UNITAU digital

ANTONIO ESIO MARCONDES SALGADO

Antonio Esio Marcondes Salgado

Linguagem de Programação I

Taubaté 2022

Autor Antonio Esio Marcondes Salgado

Unitau-Reitoria Rua Quatro de Março,432, Centro
Taubaté – São Paulo. CEP:12.020-270
Central de Atendimento:0800557255

Polo Taubaté – Sede Rua Conselheiro Moreira de Barros, 203 - Centro
Taubaté – São Paulo. CEP:12.010-080
Telefones: Coordenação Geral: (12) 3621-1530
Secretaria: (12) 3622-6050

EXPEDIENTE EDITORA

Sistema Integrado de Bibliotecas - SIBi/ UNITAU
Grupo Especial de Tratamento da Informação – GETI

Salgado, Antonio Esio Marcondes
S164e Linguagem de Programação I [recurso eletrônico] / Antonio Esio Marcondes Salgado. – Dados eletrônicos. -- Taubaté : EdUnitau, 2022.

Formato: PDF
Requisitos do sistema: Adobe
Modo de acesso: world wide web

ISBN: 978-65-86914-53-5 (on-line)

1. Linguagem de programação. 2. Orientação a objetos. 3. Java. 4. Polimorfismo. 5. Dados. I. Título.

CDD – 005.133

Ficha catalográfica elaborada pela Bibliotecária Ana Beatriz Ramos – CRB-8/6318

Índice para Catálogo sistemático

Linguagem de programação – 005.133
Orientação a objetos – 005.133
Java – 005.133
Polimorfismo – 005
Dados – 005

Sumário

Recursos de Imersão:

Explorando ideias

Eu indico

Pensando juntos

Pípulas de conhecimento

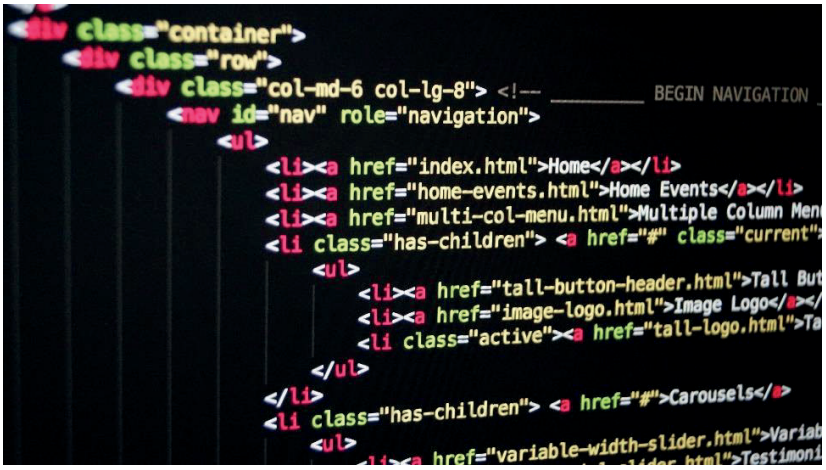
Podcast

QRCode

Linguagem de Programação I

Unidade I
Introdução à orientação a objetos

Introdução



Fonte: Pixabay

Nesta Unidade, daremos início aos estudos em orientação a objetos. Serão abordados os tópicos relacionados à abstração dos dados, a classes e objetos e atributos dos objetos.

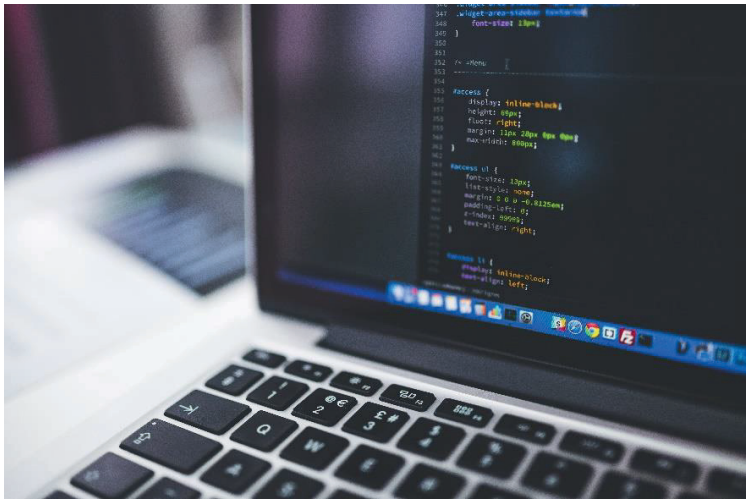
Além disso, também serão apresentados os conceitos sobre como utilizar os métodos de um objeto nas mais variadas formas de acesso e com passagens de diferentes tipos de argumentos e parâmetros.

Para se manter a segurança das informações, também serão explicados os diferentes níveis de visibilidade dos atributos e métodos de um objeto.

Todos os exemplos de linguagem de programação que serão utilizados terão como base a linguagem de programação Java, sendo ela uma das linguagens mais utilizadas nos dias de hoje e com amplo uso no mercado profissional da computação.

Bons estudos!

1.1 Conceitos e teoria de orientação a objetos



Fonte: Pixabay

O conceito de programação denominado “orientação a objetos” é um paradigma de programação (meio de classificar uma linguagem de programação) que usa como base o conceito de objetos. Um objeto pode conter elementos denominados como “campos” ou também “atributos”. Além disso, também é possível ter diversas ações que podem ser executadas por esses objetos. Essas ações são denominadas como “métodos”.

Para garantir a segurança das informações dos objetos, tanto os atributos quanto os métodos podem ter diferentes níveis de visibilidade. Esses elementos podem ser acessados somente pelo objeto ou podem ser acessados por itens fora do objeto, ficando essa questão definida de acordo com a necessidade da programação desenvolvida.

Portanto, a programação orientada a objetos tem como propósito primário aproximar o mundo da programação de computadores com o mundo real. Assim, parte do princípio de que tudo pode ser representado como objeto.

A primeira linguagem de programação com paradigma de orientação a objetos foi criada na década 1970 por Alan Kay, matemático e biólogo. Além desse importante paradigma de programação, desenvolveu diversos programas de aprendizado voltados para crianças. Sua preocupação na didática dos conceitos computacionais pode ser vista na sua frase “o computador ideal deveria funcionar como um organismo vivo, isso é, cada célula se relaciona com outras a fim de alcançar um objetivo, mas cada uma funciona de forma autônoma. As células poderiam também reagrupar-se para resolver outro problema, ou desempenhar outras funções”.

1.2 Abstração de dados, classes, objetos e atributos

Fonte: Freepik

A abstração de dados é o conceito de transformar informações do mundo real em informações adaptadas para o mundo computacional. Nesse conceito, somente as informações importantes de um objeto do mundo real são enviadas para o objeto computacional.

Assim, uma classe representa um molde de um objeto computacional. Essa classe nada mais é do que um conjunto de características (atributos) e de comportamentos (métodos) que definem o objeto representado.

Somente a classe em si é um conceito abstrato, pois ela só representa um molde. Nesse caso, é necessário que os atributos e métodos possam ser acessados. Isso ocorre através da criação de um "objeto" baseado em uma classe pré-definida. Esse processo é denominado "instanciação de classe" e, assim, a classe é usada como um molde para o "objeto". O "objeto" pode ser modificado várias vezes, conforme a necessidade da solução computacional, sendo esse processo denominado "estado de mudança".

A mudança nos objetos ocorre na variação dos valores dos "atributos". Esses valores indicam quais características um objeto pode ter e podem ser de vários tipos, tais como numéricos, texto, data/hora ou outra representação necessária para aquele objeto.

Para exemplificar todos esses conceitos, imagine um objeto de uso cotidiano na vida das pessoal, como uma caneta, por exemplo. Esse objeto, analisado da perspectiva do mundo real ou computacional, possui diversos "atributos". Para exemplificar, seguem alguns deles:

- **Cor**: indica qual a cor da tinta.
- **Tamanho**: tamanho da caneta (em centímetros).
- **Fabricante**: nome do fabricante da caneta.
- **Código de barras**: identificador único do tipo da caneta.

São atributos que podem variar de acordo com cada objeto.

De todos os atributos citados, será suposto que somente os atributos "cor" e "tamanho" serão úteis no momento. Esse critério de qual é útil ou não vai depender da análise de cada solução computacional a ser desenvolvida. Sendo assim, a classe denominada "caneta" será o molde a ser utilizado. Cada vez que um objeto novo for instanciado, a partir da classe "caneta", terá os atributos "cor" e "tamanho" e os valores desses atributos irão variar de acordo com o objeto instanciado.

1.3 Métodos, métodos estáticos ou da classe

Os métodos representam uma ação que um objeto pode realizar. A execução de um método pode acessar, modificar ou alterar os atributos de um objeto. Assim, conforme já explicado anteriormente, ocorre o "estado de mudança" de um objeto.

Os "métodos estáticos" podem ser invocados sem a criação de uma classe. São métodos especiais que podem representar trechos de processamento no qual não é necessário nem mesmo criar um objeto para acessá-lo. Um exemplo é o método "max", que retorna o maior valor de uma sequência passada para ele. A classe a que ele pertence é a classe "math".

A figura a seguir apresenta o código fonte e a respectiva saída. O método "max" é invocado na linha 5.

```
1  public class MyClass {
2      public static void main(String args[]) {
3          int valor1 = 10;
4          int valor2 = 25;
5          int maior = Math.max(valor1, valor2);
6
7          System.out.println("O maior é: " + maior);
8      }
9  }
```

```
O maior é: 25
```

Fonte: o autor (2021).

Já os "métodos de classe" só podem ser invocados se uma classe for previamente definida e posteriormente for criada uma instância de um objeto a partir dela. Assim, durante a execução do programa, essa classe será utilizada pelo objeto que representa a classe. Usando como base o exemplo da classe "caneta", os métodos de classe só podem ser invocados se um objeto for criado

a partir da classe “caneta”. Assim, os métodos disponíveis na classe “caneta” estarão disponíveis no objeto criado.

A linguagem de programação Java pode ser testada através de um compilador fornecido pelo fabricante. O site https://www.java.com/pt-BR/download/help/develop.html explica o passo a passo sobre como realizar essa ação.

Além disso, também existem compiladores *on-line* que podem ser utilizados no desenvolvimento de exemplos. Um deles pode ser acessado no site https://www.jdoodle.com/online-java-compiler/.

1.4 Sobrecarga de métodos, argumentos ou parâmetros

Conforme explicado anteriormente, os métodos representam uma ação que um objeto pode realizar. Porém, em alguns cenários, um método pode ter o mesmo nome e realizar ações diferentes. Esse processo é denominado “sobrecarga de métodos”.

A diferenciação de um método que possui o mesmo nome vai ocorrer de acordo com os parâmetros e argumentos utilizados nos parâmetros. Um exemplo disso é o método que realiza a escrita de um valor na saída padrão de um programa feito em Java. No exemplo a seguir, o método “println” tem o mesmo nome e até o mesmo comportamento, mas possuem programações internas diferentes. Na linha 18 está o método que envia para a saída padrão um conteúdo de texto e na linha 19 um conteúdo numérico.

```
1  public class MyClass {
2
3      public class Caneta(){
4
5          private String cor;
6          private double tamanho;
7
8          public void Caneta(){
9              //Cria o objeto
10         }
11     }
```

Fonte: o autor (2021)

1.5 Visibilidade de acesso

Tanto os “atributos” quanto os “métodos” dos objetos podem ter diferentes níveis de visibilidade onde eles estão inseridos. Em programação orientada a objetos, esse tipo de visibilidade é definida por uma palavra-chave. Usualmente, esses modificadores de acesso são usados para privar os atributos do acesso direto, tornando-os de acesso privado, e implementa-se métodos públicos que acessam e alteram os atributos.

Os métodos privados normalmente são usados apenas por outros métodos que são públicos e que podem ser chamados a partir de outro objeto da mesma classe. Isso ocorre para se evitar a repetição de um código em mais de um método.

Para utilizar os modificadores de acesso, é necessário que seja definido antes do nome do atributo, método ou classe do modificador. A seguir, são apresentados os principais modificadores de acesso da linguagem de programação Java:

- **Public**: deixa visível a classe ou membro para todas as outras classes, subclasses e pacotes do projeto.
- **Protected**: deixará visível o atributo para todas as outras classes e subclasses que pertençam ao mesmo pacote do projeto. A principal diferença em relação ao “public” é que apenas as classes do mesmo pacote possuem acesso ao membro. O pacote da subclasse não tem acesso ao item onde ele está inserido.

- **Private**: deixará visível o atributo somente para a classe em que esse atributo está definido.
- **Package-protected**: utilizado quando não se define nenhum dos modificadores anteriores.

Assim, o método/atributo só é visível na sua própria classe, nas classes e subclasses do mesmo pacote.

O código a seguir apresenta um exemplo de aplicação com os níveis de visibilidade explicados:

```
1  public class MyClass {
2
3      //Variável com o modificador private
4      private int numero;
5
6      //Variável com o modificador protected
7      private String nome;
8
9      //Variável com o modificador package-private
10     int codigo;
11
12     //Construtor e demais métodos
13
14     //Método com o modificador public
15     public Soma(){
16
17     }
18 }
```

Fonte: o autor (2021)

Quais as diferenças da programação estrutural para a programação orientada a objetos? Na programação estruturada, todos os métodos e objetos são visíveis todo o tempo; já na orientada a objetos, existem diferentes níveis de acesso. Além disso, por meio dos objetos há o conceito de reutilização de códigos, assim há menos código para se representar objetos.

1.6 Síntese da Unidade

Nessa Unidade, vimos os conceitos introdutórios da programação orientada a objetos. Foram abordados os conceitos de abstração de dados, a definição de classes e objetos e os seus respectivos atributos. Também foi explicado o conceito de métodos e os seus tipos (estáticos e de classe).
Por fim, foram apresentadas as sobrecargas de métodos e a visibilidade de acesso dos atributos e métodos. Junto a esses conceitos, foram também apresentadas as aplicabilidades deles, utilizando a linguagem de programação Java.

1.7 Para saber mais

Recomendamos a leitura do capítulo 6 do livro "Java: como programar", de Deitel e Deitel, publicado pela editora Prentice-Hall, para se aprofundar um pouco mais no conteúdo abordado.

1.8 Aprendendo

Diversos objetos do mundo real podem ser abstraídos e enviados para o mundo da orientação a objetos. Portanto, imagine alguns objetos que poderiam ser representados de acordo com o paradigma de orientação a objetos.

1.9 Praticando

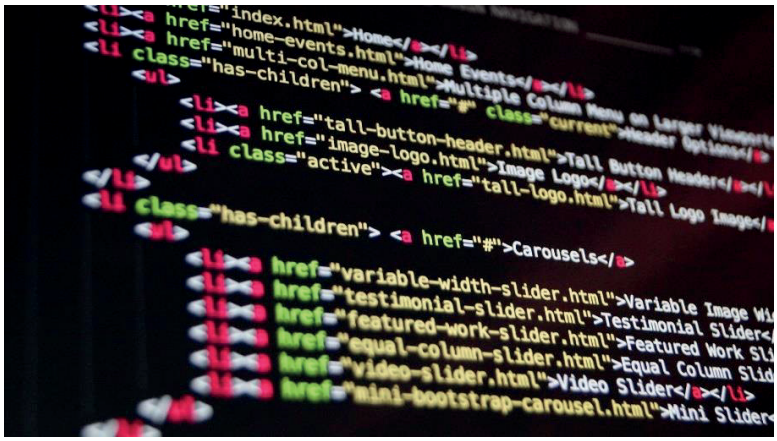
Para cada objeto criado na seção anterior, que pode ser representado no paradigma de orientação a objetos, desenvolva pelo menos cinco atributos e três métodos para eles.

1.10 Referências

DEITEL, H. M.; DEITEL, P. J. **Java:** Como Programar. Rio de Janeiro: Prentice-Hell, 2005.

Unidade II
Pilares de orientação a objetos

Introdução



Fonte: Pixabay

Nesta Unidade, daremos continuidade aos estudos em orientação a objetos. Serão abordados os tópicos relacionados a construtores, encapsulamento, herança e polimorfismo e, finalmente, a comunicação por troca de mensagens.

Tais conceitos, abordados na presente Unidade, são os pilares básicos da programação orientada a objetos e ajudarão a entender os fundamentos desse paradigma de programação.

Todos os exemplos de linguagem de programação que serão utilizados terão como base a linguagem de programação Java, por ser ela uma das linguagens mais utilizadas nos dias de hoje, com amplo uso no mercado profissional da computação.

Bons estudos!

2.1 Construtores

O método denominado “construtor” determina as ações que devem ser executadas quando um objeto é instanciado. Especificamente na linguagem de programação Java, o construtor é definido como um método cujo nome deve ser o MESMO nome da classe. Além disso, ele não deve ser um método com um tipo definido. A figura a seguir nos apresenta uma classe denominada “Caneta”, seus atributos e a definição do construtor dela.

```
3   public class Caneta{
4
5       //Atributos do campo
6       private String cor;
7       private double tamanho;
8
9       //Construtor da classe
10      public Caneta(){
11          //Código do construtor
12      }
13  }
```

Fonte: o autor (2021)

Se em uma classe nenhum construtor for definido, internamente é criado pela linguagem de programação Java um construtor automático. Por ser algo definido pela linguagem de programação Java, ele não possui nenhum tipo de parâmetro. Se a classe tiver pelo menos um construtor definido pelo desenvolvedor, o construtor automático não será definido pela linguagem de programação. A figura a seguir apresenta uma classe denominada “Caneta”, sem uma definição do construtor. Ao ser instanciada a classe adiante, internamente, a linguagem de programação Java irá criar um construtor não visível ao programador, mas para uso interno. A figura apresenta uma classe denominada “Caneta”, seus atributos e nenhum construtor foi definido pelo programador. Quando essa classe for instanciada, o construtor “Caneta” será criado internamente.

```
3   public class Caneta{
4
5       //Atributos do campo
6       private String cor;
7       private double tamanho;
8
9   }
```

Fonte: o autor (2021)

O construtor pode receber parâmetros, ação semelhante a um método. Assim é aplicado o conceito de sobrecarga, ou seja, um ou mais métodos com o mesmo nome, mas com argumentos diferentes. A figura a seguir apresenta uma classe denominada “Caneta”, seus atributos e a definição de dois construtores, sendo o primeiro construtor sem nenhum parâmetro e o segundo construtor com o parâmetro “cor”.

```
3   public class Caneta{
4
5       //Atributos do campo
6       private String cor;
7       private double tamanho;
8
9       //Construtor 1 da classe
10      public Caneta(){
11          //Código do construtor
12      }
13
14      //Construtor 2 da classe
15      public Caneta(String cor){
16          //Código do construtor
17      }
18
19  }
```

Fonte: o autor (2021)

O construtor é invocado uma única vez no momento da criação do objeto através do comando “new”. Sendo assim, o retorno do comando “new” é uma referência para o objeto criado. Assim, a partir de uma classe existente, é feita uma cópia dela para um objeto e consequentemente os métodos e atributos da classe são herdados pelo objeto recém-criado.

A figura a seguir apresenta uma classe denominada “Caneta”, seus atributos e a definição do construtor dela.

```
3   public class Caneta{
4
5       //Atributos do campo
6       private String cor;
7       private double tamanho;
8
9       //Construtor da classe
10      public Caneta(){
11          //Código do construtor
12      }
13  }
```

Fonte: o autor (2021)

A figura apresenta um objeto chamado “objetoCaneta”, que é a instância da classe “Caneta”.

```
20  public static void main(String args[]) {
21      //Instância da classe Caneta
22      //atribuída a variável objetoCaneta
23      Caneta objetoCaneta = new Caneta();
24  }
```

Fonte: o autor (2021)

Na linguagem de programação Java, todo construtor precisa ter a visibilidade “Public”, ou seja, tem que ser acessível de fora do objeto. Esse cenário é necessário, pois o construtor é invocado quando o objeto é instanciado. Assim, se ele não for público, não será possível acessar o construtor e consequentemente o objeto não seria instanciado.

2.2 Encapsulamento

O termo “encapsulamento”, em programação orientada a objetos, é um conceito que significa que os atributos ficam o mais isolados possível. Para que esse cenário ocorra, a visibilidade de acesso dos métodos e atributos pode ser manipulada.

Relembrando um conceito da Unidade anterior. Para utilizar os modificadores de acesso, é necessário que seja definido o modificador, antes do nome do atributo, método ou classe. Adiante, seguem os principais modificadores de acesso da linguagem de programação Java:

- **Public**: deixará visível a classe ou membro para todas as outras classes, subclasses e pacotes do projeto.
- **Protected**: deixará visível o atributo para todas as outras classes e subclasses que pertençam ao mesmo pacote do projeto. A principal diferença em relação ao “public” é que apenas as classes do mesmo pacote possuem acesso ao membro. O pacote da subclasse não tem acesso ao item onde ele está inserido.
- **Private**: deixará visível o atributo somente para a classe em que esse atributo está definido.
- **Package-protected**: é utilizado quando não se define nenhum dos modificadores anteriores. Assim, o método/atributo só é visível na sua própria classe, nas classes e subclasses do mesmo pacote.

O conceito do encapsulamento tem como função principal controlar o acesso aos atributos e métodos de uma classe. Assim, ele consegue determinar o que da classe pode ser acessado.
O nível de acesso com mais restrições é o modo “private”. Com isso, somente os membros internos da classe conseguem acessar o seu conteúdo. Como boa prática, recomenda-se que todos os atributos de classe sejam “Private” e para que eles possam ser manipulados (leitura e escrita) é necessário que sejam implementados métodos que executem essa ação, a qual é denominada “encapsulamento”.

Para se ter um método que faça a ação de encapsulamento, usualmente é utilizado o modificador de acesso “public”. Portando, para obter o acesso a algum atributo ou método encapsulado, utiliza-se o princípio de “get” e “set”.

Através do “get” é definido um método que retorna o valor de um atributo da classe. Esse retorno pode ser somente do atributo ou, se for o caso, manipular o conteúdo do atributo.

Já o “set” é a maneira como um atributo pode ser alterado. Através de um ou mais parâmetros no método, o atributo do campo é modificado.

O código adiante apresenta um exemplo de manipulação de atributos de um método através dos conceitos de “get” e “set”.

```
3   public class Caneta{
4
5       //Atributos do campo
6       private String cor;
7       private double tamanho;
8
9       //Construtor da classe
10      public Caneta(){
11          //Código do construtor
12      }
13
14      //Método que faz a leitura do atributo cor
15      public String getCor(){
16          return this.cor;
17      }
18
19      //Método que faz a modificação do atributo cor
20      public void setCor(String cor){
21          this.cor = cor;
22      }
23
24      //Método que faz a leitura do atributo tamanho
25      public Double getTamanho(){
26          return this.tamanho;
27      }
28
29      //Método que faz a modificação do atributo cor
30      public void setTamanho(Double tamanho){
31          this.tamanho = tamanho;
32      }
33  }
```

Fonte: o autor (2021)

No código anterior, tanto no método “get” quanto no método “set”, ao se acessar o atributo da classe, foi usado o comando “this”. Esse comando serve para indicar ao método que o atributo acessado pertente à classe e não ao método local. Isso fica mais claro ao se observar o método “GET”. O código adiante apresenta esse cenário, no qual o item “this.cor” indica que o atributo cor está sendo acessado e o item “cor” é o parâmetro do método.

```
19      //Método que faz a modificação do atributo cor
20      public void setCor(String cor){
21          this.cor = cor;
22      }
```

Fonte: o autor (2021)

O uso do comando “this” só é obrigatório se no método houver um parâmetro ou variável com o mesmo nome de um atributo da classe. Se não existir esse cenário, o atributo de classe não precisa usar o comando “this” para ser acessado. Porém, para fins de padronização e boas práticas, é recomendado o uso desse comando.

2.3 Herança

A herança é um recurso do paradigma orientação a objetos que permite a criação de novas classes a partir de uma classe já existente. Esse conceito permite que as características (atributos e métodos) de uma classe existente sejam HERDADAS pela classe estendida.

A vantagem desse recurso é que permite o conceito de reuso de um código já implementado. Além disso, caso precise ser feita alguma manutenção na classe base, automaticamente as classes estendidas terão acesso ao código atualizado.

A classe base é denominada de superclasse e as classes estendidas são também denominadas como subclasses. As subclasses são mais especializadas do que as superclasses, pois elas além de herdar as características da superclasse podem ter os seus próprios atributos e métodos.

O código adiante apresenta o conceito de “Herança”, em que a superclasse “Caneta” estende a subclasse “Lapiseira”. Na declaração da subclasse “Lapiseira” através do comando “extends” é feita a referência à superclasse “Caneta”.

```java
//Superclasse Caneta
public class Caneta{

    //Atributos do campo
    private String cor;
    private double tamanho;

    //Construtor da superclasse
    public Caneta(){
        //Código do construtor
    }

    //Método que faz a leitura do atributo cor
    public String getCor(){
        return this.cor;
    }

    //Método que faz a modificação do atributo cor
    public void setCor(String cor){
        this.cor = cor;
    }

    //Método que faz a leitura do atributo tamanho
    public Double getTamanho(){
        return this.tamanho;
    }

    //Método que faz a modificação do atributo tamanho
    public void setTamanho(Double tamanho){
        this.tamanho = tamanho;
    }
}
```

Fonte: o autor (2021)

```
36      //Subclasse Lapiseira
37      public class Lapiseira extends Caneta{
38          //Atributos do campo
39          private String tipo_grafite;
40
41          //Construtor da subclasse
42          public Lapiseira(){
43              //Código do construtor
44          }
45
46          //Método que faz a leitura do atributo tipo_grafite
47          public String getTipo_grafite(){
48              return this.tipo_grafite;
49          }
50
51          //Método que faz a modificação do atributo tipo_grafite
52          public void setTipo_grafite(String tipo_grafite){
53              this.tipo_grafite = tipo_grafite;
54          }
55      }
```

Fonte: o autor (2021)

O conceito de Herança é sempre utilizado na linguagem de programação Java. No momento em que uma classe é instanciada e não há nenhuma referência à superclasse à qual ela pertence, a classe criada herda os atributos e métodos da classe “Object”.

2.4 Polifmorfismo

O polimorfismo é um recurso do paradigma de orientação a objetos no qual duas ou mais classes derivadas de uma mesma superclasse podem utilizar métodos que têm o mesmo nome e os mesmos parâmetros, mas com comportamentos diferentes. O método de cada subclasse é codificado de acordo com a necessidade dela.

Conceitualmente, a palavra polimorfismo significa várias formas. No contexto da orientação a objetos, polimorfismo indica uma situação na qual um objeto tem a capacidade de se comportar de maneiras diferentes ao receber uma informação.

Para o polimorfismo, é obrigatório que os métodos possuam exatamente o mesmo nome de método, sendo utilizado o mecanismo de redefinição de métodos. Na linguagem de programação Java, para que a superclasse suporte o conceito de polimorfismo, na sua identificação deve ter o comando “abstract”. Além disso, cada método que deverá ser implementado nas subclasses

precisará ter na declaração do seu cabeçalho a palavra “abstract”. Depois dessas modificações na superclasse, cada subclasse será obrigada a ter implementado o(s) método(s) classificado(s) como obrigatório(s), através do comando “abstract” na superclasse.

O código adiante apresenta o conceito de “Herança”, na qual a superclasse “Caneta” estende a subclasse “Lapiseira” e a subclasse “GizCera”. Na superclasse é definido o método “escrever”, no qual é indicado que esse método deve ser implementado nas subclasses. Dentro das subclasses, foi implementado o método “escrever” no qual, para cada tipo de subclasse, é executada uma ação diferente.

O código a seguir representa a superclasse “Caneta”.

```
3       //Superclasse Caneta
4       public abstract class Caneta{
5
6           //Atributos da classe
7           private String cor;
8           private double tamanho;
9
10          //Construtor da superclasse
11          public Caneta(){
12              //Código do construtor
13          }
14
15          //Métodos da superclasse....
16
17          //Método abstrato que será implementado nas subclasses
18          public abstract void escrever(String texto);
19
20      }
```

Fonte: o autor (2021)

O código adiante representa a subclasse “Lapiseira”.

```
22      //Subclasse Lapiseira
23      public class Lapiseira extends Caneta{
24
25          //Atributos do campo
26          private String tipo_grafite;
27
28          //Construtor da subclasse
29          public Lapiseira(){
30              //Código do construtor
31          }
32
33          //Método escrever implementado para a subclasse Lapiseira
34          public void escrever(String texto){
35              System.out.println("Usando a lapiseira: " + texto);
36          }
37      }
```

Fonte: o autor (2021)

O código adiante representa a subclasse “GizCera”.

```
39      //Subclasse GizCera
40      public class GizCera extends Caneta{
41
42          //Atributos da classe
43          private double diametro;
44
45          //Construtor da subclasse
46          public GizCera(){
47              //Código do construtor
48          }
49
50          //Método escrever implementado para a subclasse GizCera
51          public void escrever(String texto){
52              System.out.println("Usando o giz de cera: " + texto);
53          }
54      }
```

Fonte: o autor (2021)

O conceito de polimorfismo não deve ser confundido com o mecanismo de sobrecarga de métodos. Na sobrecarga de métodos, a diferenciação de um método, que possua o mesmo nome, vai ocorrer de acordo com os parâmetros e argumentos utilizados nos parâmetros.

2.5 Comunicação por troca de mensagens

Os objetos de um código desenvolvido utilizando o paradigma de orientação a objetos estão sempre se comunicando para realizar alguma atividade.

Esse processo de comunicação do objeto é feito através de um outro processo denominado troca de mensagens.

Nesse cenário, cada mensagem acaba sendo requisição para que um objeto execute uma operação específica.

Essas operações podem ser acessos aos métodos ou atributos de uma classe, desde que eles estejam com a visibilidade como “public”. Se o acesso for a um atributo, basta invocar o nome do objeto instanciado e o nome do atributo, conforme o exemplo adiante: “Objeto.Atributo”. Se o acesso for a um método, será um cenário semelhante ao acesso do método: deverão ser invocados o objeto e o método, conforme o exemplo “Objeto.Método”.

O código a seguir apresenta uma classe chamada “Caneta” e como ocorre a troca de mensagens dentro dela.

A classe “Caneta” possui atributos e métodos. Para cada atributo, foi definido um método “get”, responsável por retornar o valor de um atributo, e um método “set”, que vai permitir a alteração de um atributo. Além desses métodos, manipuladores de métodos, foram definidos mais dois métodos que vão fazer funções específicas para a classe caneta.

```
3       public class Caneta{
4
5           //Atributos do campo
6           private String cor;
7           private double tamanho;
8
9           //Construtor da classe
10          public Caneta(){
11              //Código do construtor
12          }
13
14          //Método que faz a leitura do atributo cor
15          public String getCor(){
16              return this.cor;
17          }
18
19          //Método que faz a modificação do atributo cor
20          public void setCor(String cor){
21              this.cor = cor;
22          }
23
24          //Método que faz a leitura do atributo tamanho
25          public Double getTamanho(){
26              return this.tamanho;
27          }
28
29          //Método que faz a modificação do atributo cor
30          public void setTamanho(Double tamanho){
31              this.tamanho = tamanho;
32          }
33
34          //Método que escreve na saída padrão o texto passado por parâmetro
35          public void escrever(String texto){
36              System.out.println(texto);
37          }
38      }
```

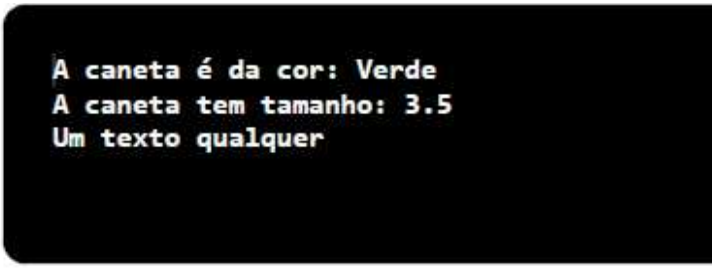
Fonte: o autor (2021)

A classe “main” vai criar uma instância da classe “Caneta”. O objeto que vai receber essa instância é o “objetoCaneta”. Depois dessa ação, os métodos e atributos públicos estarão disponíveis para serem acessados. Uma ressalva: todos os atributos do objeto estão como privados, a única forma de se manipular esses atributos é através dos comandos “get” e “set”. Além disso, também existe um método chamado “escrever”, que imprime na saída padrão do computador um texto informado pelo programador através de um parâmetro do tipo String.

```
41      public static void main(String args[]) {
42          //Instância da clase Caneta no item objetoCaneta
43          Caneta objetoCaneta = new Caneta();
44
45          //Efetuando a comunicação de mensagem
46
47          //Alterando os atributos da classe através dos métodos do tipo "set"
48          objetoCaneta.setCor("Verde");
49          objetoCaneta.setTamanho(3.5);
50
51          //Retornando os atributos da classe através dos métodos do tipo "get"
52          System.out.println("A caneta é da cor: " + objetoCaneta.getCor());
53          System.out.println("A caneta tem tamanho: " + objetoCaneta.getTamanho());
54
55          //Executando um método que efetua alguma ação dentro da classe
56          objetoCaneta.escrever("Um texto qualquer");
57      }
```

Fonte: o autor (2021)

A saída da execução da classe “main” pode ser observada na figura a seguir:



Fonte: o autor (2021)

2.6 Síntese da Unidade

Nesta Unidade, vimos os pilares básicos da orientação referente ao encapsulamento dos atributos e métodos, herança e polimorfismo das classes.

Juntamente a esses conceitos, foram apresentadas as aplicabilidades deles utilizando a linguagem de programação Java.

Além disso, pelo fato de ter sido aqui abordada a linguagem de programação Java, foram explicados conceitos específicos dessa linguagem no paradigma de orientação a objetos.

2.7 Para saber mais

Recomendamos a leitura do Capítulo 6 do livro “Java: como programar”, dos autores Deitel e Deitel, da editora Prentice-Hall, para se aprofundar um pouco mais no conteúdo abordado.

2.8 Aprendendo

Diversos objetos do mundo real podem ser abstraídos e enviados para o mundo da orientação a objetos. Portanto, imagine alguns objetos que poderiam ser representados de acordo com o paradigma de orientação a objetos.

2.9 Praticando

Para cada objeto criado na seção anterior, que pode ser representado no paradigma de orientação a objetos, desenvolva pelo menos cinco atributos e três métodos, utilizando a linguagem de programação Java.

Posteriormente, desenvolva classes na linguagem de programação Java utilizando os conceitos de herança e polimorfismo.

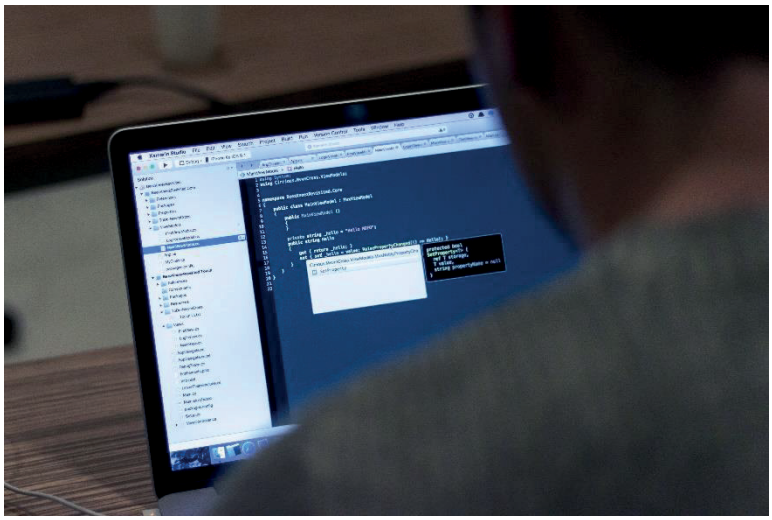
Por fim, crie instâncias das classes desenvolvidas e invoque os seus respectivos atributos e métodos.

2.10 Referências

DEITEL, H. M.; DEITEL, P. J. **Java:** Como Programar. Rio de Janeiro: Prentice-Hell, 2005.

Unidade III
Aplicação dos pilares da orientação a objetos – Parte 1

Introdução



Fonte: Pixabay

Nesta Unidade, trataremos sobre a aplicação dos conceitos explicados nas Unidades anteriores, nos assuntos relacionados à orientação a objetos. Serão abordados os tópicos de construtores, encapsulamento e herança.

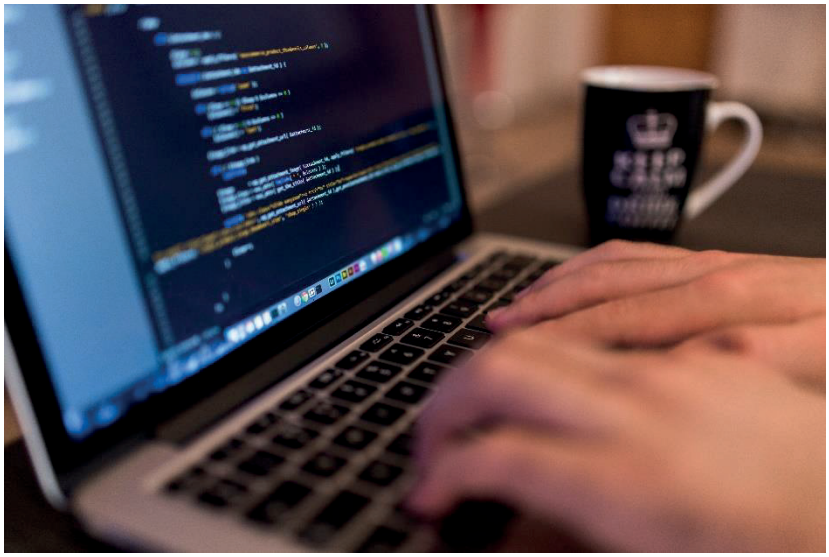
Todos os exemplos de linguagem de programação que serão utilizados terão como base a linguagem de programação Java, sendo ela uma das linguagens mais utilizadas nos dias de hoje e com amplo uso no mercado profissional da computação.

Bons estudos!

3.1 Definição de uma classe de exemplo

Para aplicar os conceitos apresentados nos capítulos anteriores, será definida uma classe que representa um animal. Analisando do ponto de vista de orientação a objetos, será definida uma classe para representar este animal. Nela, serão definidos alguns atributos e métodos. Além disso, depois que a classe base estiver montada, para entender outros conceitos, como herança e polimorfismo, serão criadas subclasses que vão representar as extensões da classe animal.

3.2 Definição da classe base



Fonte: Pixabay

Conforme descrito no tópico anterior, a classe Animal vai representar um animal e terá os atributos cor, peso e temSom. Além disso, também foi definido o construtor básico da classe. O código adiante apresenta o código dessa classe.

```
1   public class Animal {
2
3       //Declarando os atributos
4       private String cor;
5       private double peso;
6       private boolean temSom;
7
8       //Construtor padrão
9       public Animal(){
10          //Código do construtor
11      }
12
13  }
14
```

Fonte: o autor (2021).

Observe que todos os atributos estão no modo “private”, ou seja, só são visíveis dentro da classe Animal. Já o construtor está no modo “Package-protected”, ou seja, só é visível na sua própria classe, nas classes e subclasses do mesmo pacote. Além disso, o construtor da classe também foi definido sem nenhum código.

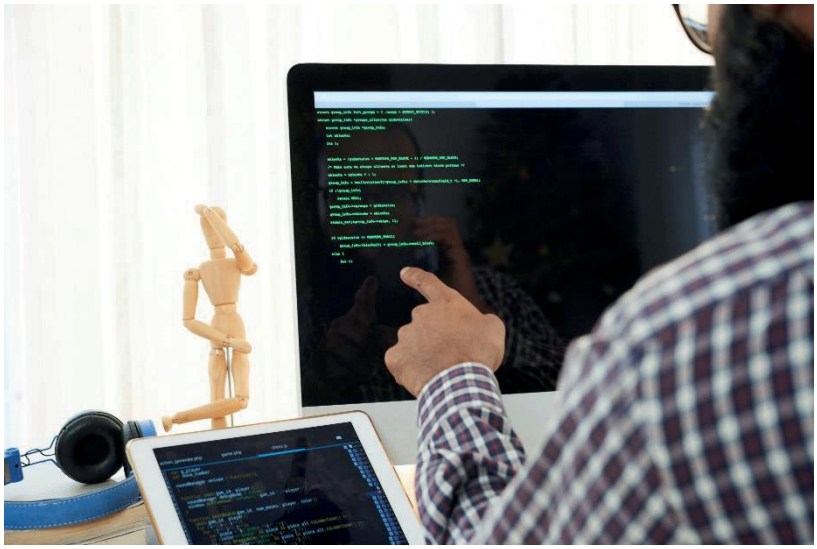
Podem ser definidos diversos tipos de construtores, desde que os parâmetros do construtor sejam diferentes. O código a seguir tem um exemplo da classe Animal com mais de um construtor. No segundo construtor, são definidos valores iniciais para os atributos da classe. No terceiro construtor são enviados nos parâmetros os valores para os atributos cor e peso; já para o atributo temSom é definido um valor como “false”.

```
1   public class Animal {
2
3       //Declarando os atributos
4       private String cor;
5       private double peso;
6       private boolean temSom;
7
8       //Construtor padrão
9       public Animal(){
10          //Código do construtor
11      }
12
13      //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
14      public Animal(String cor, double peso, boolean temSom){
15          //Código do construtor
16          this.cor = cor;
17          this.peso = peso;
18          this.temSom = temSom;
19      }
20
21      //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
22      public Animal(String cor, double peso, boolean temSom){
23          //Código do construtor
24          this.cor = cor;
25          this.peso = peso;
26          this.temSom = temSom;
27      }
28
29      //Construtor com dois atributos obrigatórios, no caso cor e peso
30      //O atributo temSom foi definido com o valor "false"
31      public Animal(String cor, double peso){
32          this.setCor(cor);
33          this.setPeso(peso);
34          this.setTemSom(false);
35      }
36
37  }
```

Fonte: o autor (2021).

O código anterior deve ser salvo em uma pasta da sua preferência com o nome “Animal.java”.

3.3 Encapsulamento dos atributos



Fonte: Freepik

Conforme os códigos apresentados no tópico anterior, os atributos foram inicializados a partir do segundo construtor com parâmetros, porém essa não é considerada uma boa prática. O ideal é que existam métodos, do tipo público, para se alterar os atributos que são do tipo privado. Além da questão de boa prática, existe a questão de segurança dos atributos, pois se eles foram do tipo público serão acessados de uma forma direta pela instância da classe.

Dessa forma, o código a seguir apresenta a classe Animal com todos os seus atributos devidamente encapsulados através dos métodos “Get” e “Set”.

```java
public class Animal {

    //Declarando os atributos
    private String cor;
    private double peso;
    private boolean temSom;

    //Construtor padrão
    public Animal(){
        //Valores padrão de inicialização
        this.setCor("");
        this.setPeso(0);
        this.setTemSom(false);
    }

    //Construtor com dois atributos obrigatórios, no caso cor e peso
    //O atributo temSom foi definido com o valor "false"
    public Animal(String cor, double peso){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(false);
    }

    //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
    public Animal(String cor, double peso, boolean temSom){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);
    }

    public boolean getTemSom() {
        return temSom;
    }

    public void setTemSom(boolean temSom) {
        this.temSom = temSom;
    }

    public String getCor() {
        return this.cor;
    }

    public void setCor(String cor) {
        this.cor = cor;
    }

    public double getPeso() {
        return this.peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }
}
```

Fonte: o autor (2021).

O código anterior deve ser salvo em uma pasta da sua preferência com o nome “Animal.java”.

3.4 Herança

Fonte: Pixabay

Para relembrar, o conceito de herança é um recurso do paradigma orientação a objetos que permite a criação de novas classes a partir de uma classe já existente. Esse conceito permite que as características (atributos e métodos) de uma classe existente (superclasse) sejam HERDADAS pela classe estendida (subclasse).

A classe base é denominada de superclasse e as classes estendidas são também denominadas como subclasses. As subclasses são mais especializadas do que as superclasses, pois elas além de herdar as características da superclasse podem ter os seus próprios atributos e métodos.

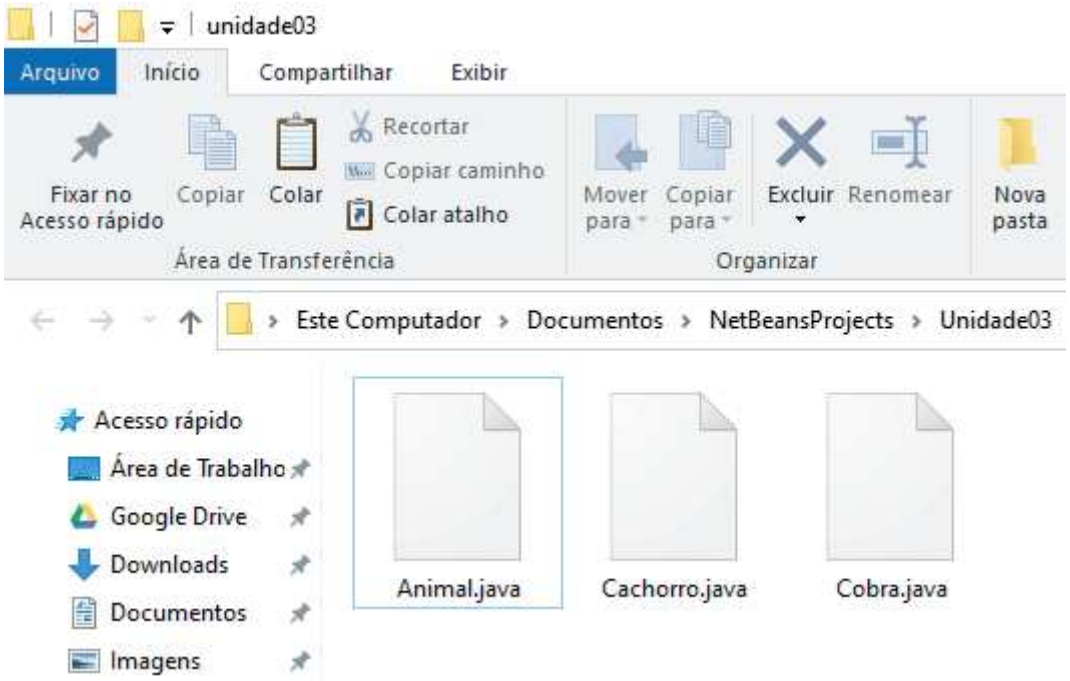
Dando continuidade ao desenvolvimento do projeto de exemplo da Unidade, a superclasse será a Animal. Lembrando que esse conteúdo deve ser salvo em uma pasta da sua preferência com o nome Animal.java. O código adiante apresenta a classe Animal com todos os seus atributos devidamente encapsulados.

```java
public class Animal {

    //Declarando os atributos
    private String cor;
    private double peso;
    private boolean temSom;

    //Construtor padrão
    public Animal(){
        //Valores padrão de inicialização
        this.setCor("");
        this.setPeso(0);
        this.setTemSom(false);
    }

    //Construtor com dois atributos obrigatórios, no caso cor e peso
    //O atributo temSom foi definido com o valor "false"
    public Animal(String cor, double peso){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(false);
    }

    //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
    public Animal(String cor, double peso, boolean temSom){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);
    }

    public boolean getTemSom() {
        return temSom;
    }

    public void setTemSom(boolean temSom) {
        this.temSom = temSom;
    }

    public String getCor() {
        return this.cor;
    }

    public void setCor(String cor) {
        this.cor = cor;
    }

    public double getPeso() {
        return this.peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }
}
```

Fonte: o autor (2021).

Para exemplificar, serão definidas duas classes estendidas. Uma subclasse será Cachorro e a outra será Cobra. Cada uma delas terá atributos e métodos específicos para cada um dos animais. Todavia, elas terão como classe base Animal, na qual todos os atributos e métodos definidos serão herdados pelas subclasses. Para cada subclasse deverá ser criado um arquivo com os seguintes nomes: Cachorro.java e Cobra.java. Esses arquivos devem ser salvos na

mesma pasta da classe Animal.java. A figura a seguir demonstra a estrutura desses arquivos no Windows Explorer.



Fonte: o autor (2021).

Para se manter uma melhor didática, será desenvolvida a classe “Cachorro.java” passo a passo. Depois que ela for desenvolvida, será apresentada a classe “Cobra.java” totalmente desenvolvida.

Na linguagem de programação Java é bem simples aplicar o conceito de herança nas classes filhas. Para que ocorra essa identificação, no código de identificação da classe filha deverá ser acrescentada a palavra-chave “extends” e ser indicada qual é a classe filha, no caso Animal. No trecho de código adiante, está a declaração da classe “Cachorro.java” que estende da classe “Animal.java” e ainda sem ter nenhum atributo, construtor ou método definido.

```
1   public class Cachorro extends Animal{
2       //Atributos
3
4       //Construtores
5
6       //Métodos
7   }
8
```

Fonte: o autor (2021).

A classe “Cachorro.java”, pelo fato de a classe “Animal.java” já ser estendida a ela, herdará automaticamente todos os atributos e métodos da classe base.

A classe “Cachorro.java” tem atributos e métodos que são específicos dos cachorros. Assim, foram definidos os seguintes atributos e métodos específicos da classe estendida.

```
1   public class Cachorro extends Animal{
2
3       //Declarando os atributos
4       private String raca;
5       private boolean pedigree;
6
7       public String getRaca() {
8           return raca;
9       }
10
11      public void setRaca(String raca) {
12          this.raca = raca;
13      }
14
15      public boolean getPedigree() {
16          return pedigree;
17      }
18
19      public void setPedigree(boolean pedigree) {
20          this.pedigree = pedigree;
21      }
22
23  }
24
```

Fonte: o autor (2021).

O código anterior apresenta os atributos específicos de um cachorro e os métodos que encapsulam esses atributos. Além desse código, agora será apresentado também um construtor para a subclasse. Observe que esse construtor altera os valores da superclasse. Essas alterações de atributos da superclasse ocorrem através da palavra “this”. Quando ela é invocada, os métodos e atributos da superclasse são disponibilizados para a classe estendida. Lembrando que a visibilidade desses elementos respeita os modificadores “Public” “Private” e afins da linguagem de programação Java. O código a seguir apresenta a classe “Cachorro.java” com o construtor devidamente definido e os atributos e métodos também devidamente codificados.

```java
public class Cachorro extends Animal{

    //Declarando os atributos
    private String raca;
    private boolean pedigree;

    //Construtor padrão: Obrigando a preencher os atributos raca e pedigree
    public Cachorro(String cor, double peso, boolean temSom, String raca, boolean
    pedigree){
        //Preenchendo atributos da classe Animal
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);

        //Preenchendo atributos da classe Cachorro
        this.setRaca(raca);
        this.setPedigree(pedigree);
    }

    public String getRaca() {
        return raca;
    }

    public void setRaca(String raca) {
        this.raca = raca;
    }

    public boolean isPedigree() {
        return pedigree;
    }

    public void setPedigree(boolean pedigree) {
        this.pedigree = pedigree;
    }

}
```

Fonte: o autor (2021).

Portanto, foi apresentado o conceito de herança para o cenário no qual tem uma superclasse (Animal.java) e uma subclasse (Cachorro.java). Para fixar o conteúdo, é apresentada a seguir mais uma classe estendida da superclasse “Animal.java”. É a subclasse “Cobra.java”, semelhante à subclasse anterior, que possui os atributos e métodos específicos de uma cobra.

```java
public class Cobra extends Animal{
    //Declarando os atributos específicos da classe estendida.
    private String especie;
    private boolean venenosa;

    //Construtor padrão: Obrigando a preencher os atributos raca e pedigree
    public Cobra(String cor, double peso, boolean temSom, String especie, boolean
    venenosa){
        //Preenchendo atributos da classe Animal
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);

        //Preenchendo atributos da classe Cobra.java
        this.setEspecie(especie);
        this.setVenenosa(venenosa);
    }

    public String getEspecie() {
        return especie;
    }

    public void setEspecie(String especie) {
        this.especie = especie;
    }

    public boolean getVenenosa() {
        return venenosa;
    }

    public void setVenenosa(boolean venenosa) {
        this.venenosa = venenosa;
    }

}
```

Fonte: o autor (2021).

3.5 Síntese da Unidade

Nesta Unidade, vimos a aplicação dos conceitos de construtores, encapsulamento e de herança com linguagem de programação.

Além disso, como foi utilizada como linguagem de programação Java, foram explicados conceitos específicos dessa linguagem no paradigma de orientação a objetos.

3.6 Para saber mais

Recomendamos que seja feita a leitura do capítulo 6 do livro “Java: como programar”, dos autores Deitel e Deitel, da editora Prentice-Hall, para se aprofundar um pouco mais no conteúdo abordado.

3.7 Aprendendo

Diversos objetos do mundo real podem ser abstraídos e enviados para o mundo da orientação a objetos. Portanto, imagine alguns objetos que poderiam ser representados de acordo com o paradigma de orientação a objetos.

3.8 Praticando

Imagine objetos que possam representar os conceitos aplicados nesta Unidade. Posteriormente, desenvolva classes na linguagem de programação Java utilizando os conceitos de encapsulamento, construtor e de herança.

3.9 Referências

DEITEL, H. M.; DEITEL, P. J. **Java:** Como Programar. Rio de Janeiro: Prentice-Hell, 2005.

Unidade IV
Aplicação dos pilares da orientação a objetos - Parte II

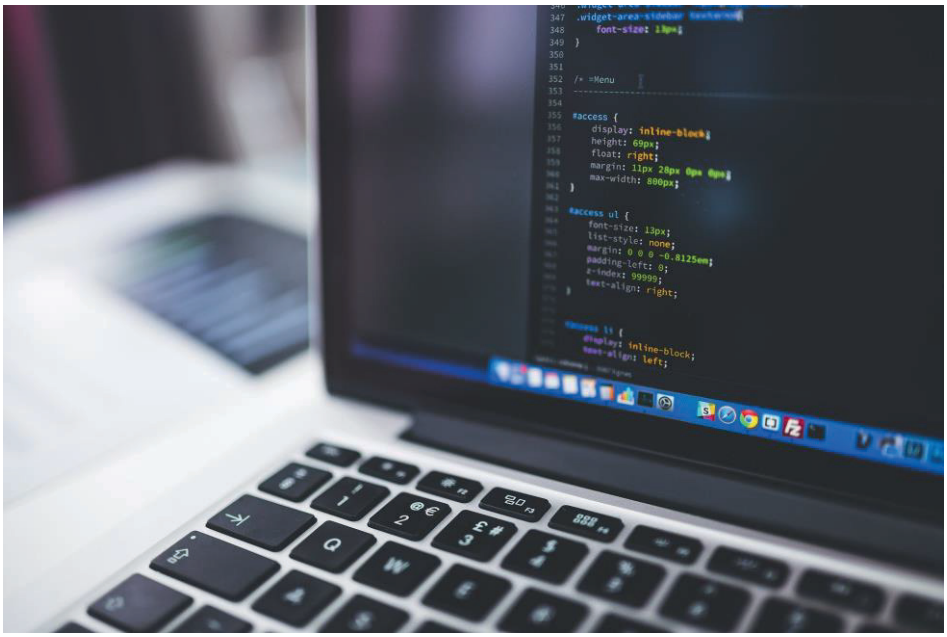
Introdução

Nesta Unidade, faremos a aplicação dos conceitos explicados nas Unidades anteriores, nos assuntos relacionados à orientação a objetos. Serão abordados os tópicos de construtores, encapsulamento, herança, polimorfismo e comunicação por troca de mensagens.

Todos os exemplos de linguagem de programação que serão utilizados terão como base a linguagem de programação Java, sendo ela uma das linguagens mais utilizadas nos dias de hoje e de amplo uso no mercado profissional da computação.

Bons estudos!

4.1 Definição de uma classe de exemplo



Fonte: Pixabay

Para aplicar os conceitos apresentados nos capítulos anteriores, será definida uma classe que representa um animal. Analisando do ponto de vista de orientação a objetos, será definida uma classe que representa esse animal. Nela serão definidos alguns atributos e métodos. Além disso, depois que a classe base estiver montada, para entender outros conceitos, tais como herança, polimorfismo e a troca de mensagens, serão criadas subclasses que vão representar as extensões da classe animal.

4.2 Relembrando a definição da classe base

Os itens explicados nesse capítulo são os mesmos da classe base da Unidade anterior. Conforme descrito na seção anterior, a classe Animal vai representar um animal e terá os atributos cor, peso e temSom. Além disso, também foi definido o construtor básico da classe. O código a seguir apresenta o código dessa classe.

```
1   public class Animal {
2
3       //Declarando os atributos
4       private String cor;
5       private double peso;
6       private boolean temSom;
7
8       //Construtor padrão
9       public Animal(){
10          //Código do construtor
11      }
12
13  }
14
```

Fonte: o autor (2021).

Observe que todos os atributos estão no modo “private”, ou seja, só são visíveis dentro da classe Animal. Já o construtor está no modo “Package-protected”, portanto, só é visível na sua própria classe, nas classes e subclasses do mesmo pacote. Além disso, o construtor da classe também foi definido sem nenhum código.

Podem ser definidos diversos tipos de construtores, desde que os parâmetros do construtor sejam diferentes. O código a seguir tem um exemplo da classe Animal com mais de um construtor. No segundo construtor, são definidos valores iniciais para os atributos da classe. No terceiro construtor, são enviados nos parâmetros os valores para os atributos cor e peso, já para o atributo temSom é definido um valor como “false”.

```java
public class Animal {

    //Declarando os atributos
    private String cor;
    private double peso;
    private boolean temSom;

    //Construtor padrão
    public Animal(){
        //Código do construtor
    }

    //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
    public Animal(String cor, double peso, boolean temSom){
        //Código do construtor
        this.cor = cor;
        this.peso = peso;
        this.temSom = temSom;
    }

    //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
    public Animal(String cor, double peso, boolean temSom){
        //Código do construtor
        this.cor = cor;
        this.peso = peso;
        this.temSom = temSom;
    }

    //Construtor com dois atributos obrigatórios, no caso cor e peso
    //O atributo temSom foi definido com o valor "false"
    public Animal(String cor, double peso){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(false);
    }

}
```

Fonte: o autor (2021).

O código anterior deve ser salvo em uma pasta da sua preferência com o nome “Animal.java”.

4.3 Relembrando o encapsulamento dos atributos

Fonte: Pixabay

Os itens explicados nesse capítulo são os mesmos da classe base da Unidade anterior. Conforme os códigos apresentados na seção anterior, os atributos foram inicializados através do segundo construtor com parâmetros, porém essa não é uma boa prática. O ideal é que existam métodos, do tipo público, para se alterar os atributos que são do tipo privado. Além da questão de boa prática, existe a questão de segurança dos atributos, pois se eles forem do tipo público serão acessados de uma forma direta pela instância da classe.

Dessa forma, o código a seguir apresenta a classe Animal com todos os seus atributos devidamente encapsulados através dos métodos “Get” e “Set”.

```java
public class Animal {

    //Declarando os atributos
    private String cor;
    private double peso;
    private boolean temSom;

    //Construtor padrão
    public Animal(){
        //Valores padrão de inicialização
        this.setCor("");
        this.setPeso(0);
        this.setTemSom(false);
    }

    //Construtor com dois atributos obrigatórios, no caso cor e peso
    //O atributo temSom foi definido com o valor "false"
    public Animal(String cor, double peso){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(false);
    }

    //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
    public Animal(String cor, double peso, boolean temSom){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);
    }

    public boolean getTemSom() {
        return temSom;
    }

    public void setTemSom(boolean temSom) {
        this.temSom = temSom;
    }

    public String getCor() {
        return this.cor;
    }

    public void setCor(String cor) {
        this.cor = cor;
    }

    public double getPeso() {
        return this.peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }
}
```

Fonte: o autor (2021).

O código anterior deve ser salvo em uma pasta da sua preferência com o nome “Animal.java”.

4.4 Polimorfismo

Para relembrar, o polimorfismo é um recurso do paradigma orientação a objetos no qual duas ou mais classes derivadas de uma mesma superclasse podem utilizar métodos que têm o mesmo nome e os mesmos parâmetros, mas com comportamentos diferentes, sendo que o método de cada subclasse é codificado de acordo com a necessidade dela.

Conceitualmente, a palavra polimorfismo significa várias formas. No contexto da orientação a objetos, polimorfismo indica uma situação na qual um objeto tem a capacidade de se comportar de maneiras diferentes ao receber uma informação.

Para o polimorfismo, é obrigatório que os métodos possuam exatamente o mesmo nome de método, sendo utilizado o mecanismo de redefinição de métodos. Na linguagem de programação Java, para que a superclasse dê suporte ao conceito de polimorfismo, na sua identificação deve ter o comando "abstract". Além disso, cada método que a ser implementado nas subclasses deverá ter na declaração do seu cabeçalho a palavra "abstract". Depois dessas modificações na superclasse, cada subclasse será obrigada a ter implementado o(s) método(s) classificados como obrigatórios, através do comando "abstract" na superclasse.

Dando continuidade ao desenvolvimento do projeto de exemplo da Unidade, a superclasse será a Animal. Vale lembrar que esse conteúdo deve ser salvo em uma pasta da sua preferência com o nome Animal.java. O código a seguir apresenta a classe Animal com todos os seus atributos devidamente encapsulados e na linha 56 está a linha na qual o método abstrato é declarado; esse método deverá ser implementado nas subclasses. Como o código é extenso, ele foi dividido em duas partes:

```java
public abstract class Animal {

    //Declarando os atributos
    private String cor;
    private double peso;
    private boolean temSom;

    //Construtor padrão
    public Animal(){
        //Valores padrão de inicialização
        this.setCor("");
        this.setPeso(0);
        this.setTemSom(false);
    }

    //Construtor com dois atributos obrigatórios, no caso cor e peso
    //O atributo temSom foi definido com o valor "false"
    public Animal(String cor, double peso){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(false);
    }

    //Construtor com três atributos obrigatórios, no caso cor, peso e temSom
    public Animal(String cor, double peso, boolean temSom){
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);
    }
```

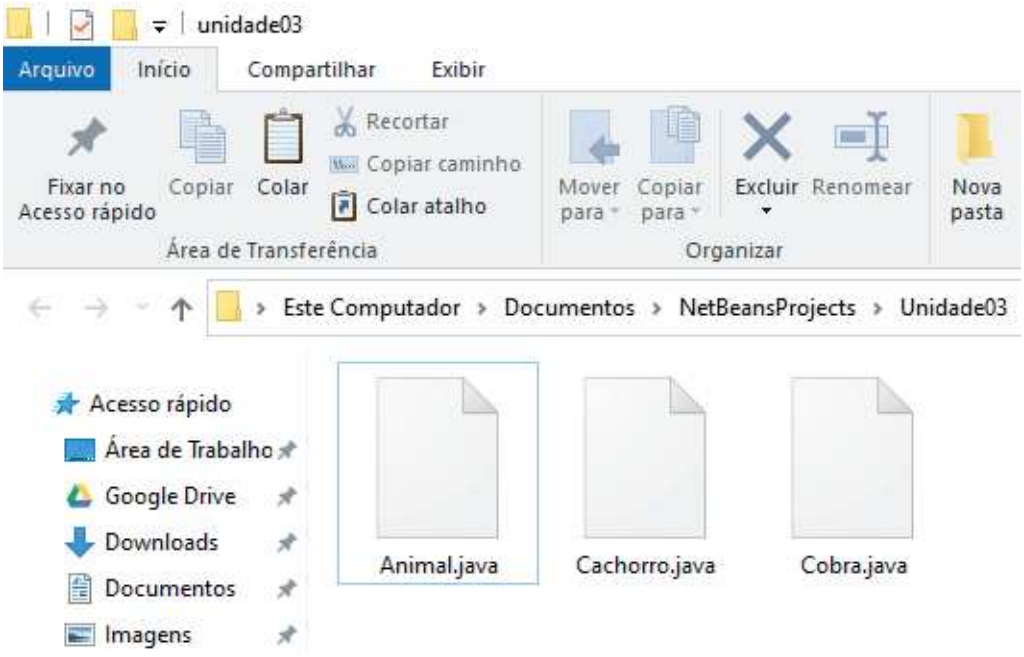
Fonte: o autor (2021).

```java
    public boolean getTemSom() {
        return temSom;
    }

    public void setTemSom(boolean temSom) {
        this.temSom = temSom;
    }

    public String getCor() {
        return this.cor;
    }

    public void setCor(String cor) {
        this.cor = cor;
    }

    public double getPeso() {
        return this.peso;
    }

    public void setPeso(double peso) {
        this.peso = peso;
    }

    //Método abstrato que será implementado nas subclasses
    public abstract void emitir_som();
}
```

Fonte: o autor (2021).

Para exemplificar, serão definidas duas classes estendidas. Uma subclasse será Cachorro e a outra será Cobra. Cada uma delas terá atributos e métodos específicos para cada um dos animais. Todavia, elas terão como classe base Animal e todos os atributos e métodos nela definidos serão herdados pelas subclasses. Além disso, cada subclasse teve implementado o método abstrato “emitir_som”.

Para cada subclasse deverá ser criado um arquivo com os seguintes nomes: Cachorro.java e Cobra.java. Esses arquivos devem ser salvos na mesma pasta da classe Animal.java. A figura a seguir demonstra a estrutura desses arquivos no Windows Explorer.



Fonte: o autor (2021).

Para se manter uma melhor didática, será desenvolvida a classe “Cachorro.java” passo a passo. Depois que ela for desenvolvida, será apresentada a classe “Cobra.java” totalmente desenvolvida.

Na linguagem de programação Java é bem simples se aplicar o conceito de polimorfismo nas classes filhas. Porém, antes disso, deverá ser implementado o conceito de herança. Para que ocorra essa identificação, no código de identificação da classe filha deverá ser acrescentada a palavra-chave “extends” e ser indicada qual é a classe filha, no caso Animal. No trecho de código a seguir está a declaração da classe “Cachorro.java” que estende da classe “Animal.java” e ainda sem ter nenhum atributo, construtor ou método definido.

```
1   public class Cachorro extends Animal{
2       //Atributos
3
4       //Construtores
5
6       //Métodos
7   }
8
```

Fonte: o autor (2021).

A classe “Cachorro.java”, por conta de a classe “Animal.java” já ser estendida a ela, herdará automaticamente todos os atributos e métodos da classe base.

A classe “Cachorro.java” tem atributos e métodos que são específicos dos cachorros. Assim, foram definidos os atributos e métodos específicos da classe estendida. O código adiante apresenta a classe “Cachorro.java” com o construtor devidamente definido e os atributos e métodos também devidamente codificados.

```
1   public class Cachorro extends Animal{
2
3       //Declarando os atributos
4       private String raca;
5       private boolean pedigree;
6
7       //Construtor padrão: Obrigando a preencher os atributos raca e pedigree
8       public Cachorro(String cor, double peso, boolean temSom, String raca, boolean
        pedigree){
9           //Preenchendo atributos da classe Animal
10          this.setCor(cor);
11          this.setPeso(peso);
12          this.setTemSom(temSom);
13
14          //Preenchendo atributos da classe Cachorro
15          this.setRaca(raca);
16          this.setPedigree(pedigree);
17      }
18
19      public String getRaca() {
20          return raca;
21      }
22
23      public void setRaca(String raca) {
24          this.raca = raca;
25      }
26
27      public boolean isPedigree() {
28          return pedigree;
29      }
30
31      public void setPedigree(boolean pedigree) {
32          this.pedigree = pedigree;
33      }
34
35  }
```

Fonte: o autor (2021).

Com a classe estendida “Cachorro.java” devidamente codificada, será implementado o método abstrato que foi definido na classe base. Só para relembrar, o nome do método é “emitir_som”. No código a seguir, a partir da linha 36, é apresentada a implementação do método abstrato. Como o código é extenso, ele foi dividido em duas partes:

```java
public class Cachorro extends Animal{

    //Declarando os atributos
    private String raca;
    private boolean pedigree;

    //Construtor padrão: Obrigando a preencher os atributos raca e pedigree
    public Cachorro(String cor, double peso, boolean temSom, String raca, boolean
    pedigree){
        //Preenchendo atributos da classe Animal
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);

        //Preenchendo atributos da classe Cachorro
        this.setRaca(raca);
        this.setPedigree(pedigree);
    }

    public String getRaca() {
        return raca;
    }
```

Fonte: o autor (2021).

```java

    public void setRaca(String raca) {
        this.raca = raca;
    }

    public boolean isPedigree() {
        return pedigree;
    }

    public void setPedigree(boolean pedigree) {
        this.pedigree = pedigree;
    }

    //Implementação do método "emitir_som". Ele está definido na super classe
    public void emitir_som() {
        if (this.getTemSom() == true) {
            System.out.println("Auau");
        }
        else {
            System.out.println("Animal sem som");
        }

    }
}
```

Fonte: o autor (2021).

Portanto, foi apresentado o conceito de polimorfismo para o cenário no qual tem uma superclasse (Animal.java) e uma subclasse (Cachorro.java). Para fixar o conteúdo, é apresentada a seguir mais uma classe estendida da superclasse “Animal.java”. É a subclasse “Cobra.java”, semelhante à subclasse anterior, que possui a implementação do método abstrato “emitir_som”. Como o código é extenso, ele foi dividido em duas partes:

```java
public class Cobra extends Animal{
    //Declarando os atributos específicos da classe estendida.
    private String especie;
    private boolean venenosa;

    //Construtor padrão: Obrigando a preencher os atributos raca e pedigree
    public Cobra(String cor, double peso, boolean temSom, String especie, boolean
    venenosa){
        //Preenchendo atributos da classe Animal
        this.setCor(cor);
        this.setPeso(peso);
        this.setTemSom(temSom);

        //Preenchendo atributos da classe Cobra.java
        this.setEspecie(especie);
        this.setVenenosa(venenosa);
    }

    public String getEspecie() {
        return especie;
    }

```

Fonte: o autor (2021).

```java
    public void setEspecie(String especie) {
        this.especie = especie;
    }

    public boolean getVenenosa() {
        return venenosa;
    }

    public void setVenenosa(boolean venenosa) {
        this.venenosa = venenosa;
    }

    //Implementação do método "emitir_som". Ele está definido na super classe
    public void emitir_som() {
        if (this.getTemSom() == true) {
            System.out.println("shi");
        }
        else {
            System.out.println("Animal sem som");
        }

    }

}
```

Fonte: o autor (2021).

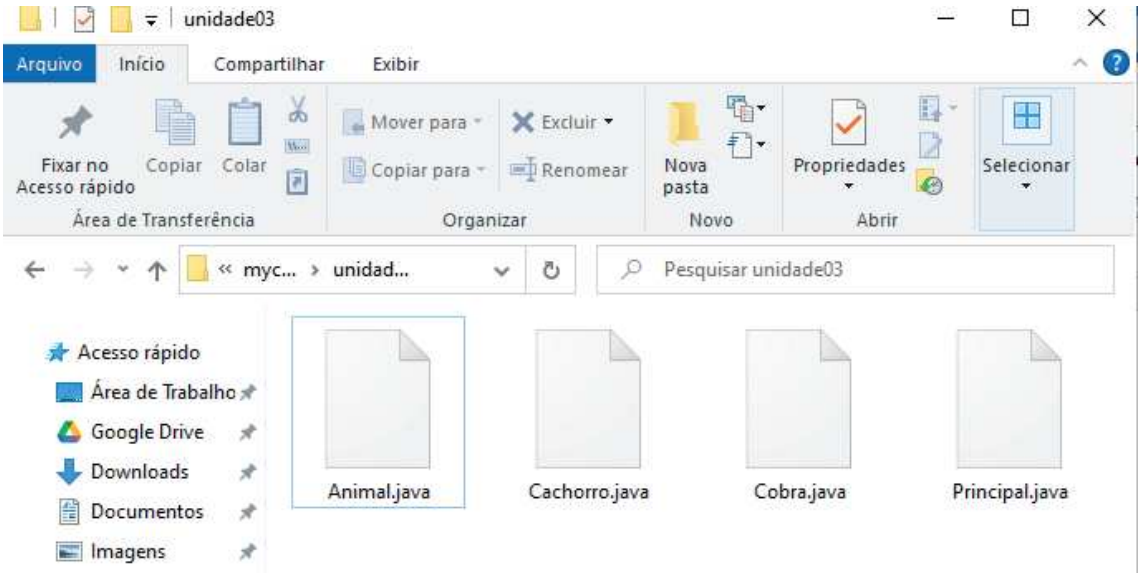
4.5 Comunicação por troca de mensagens

Para relembrar, os objetos de um código desenvolvido utilizando o paradigma de orientação a objetos estão sempre se comunicando para realizar alguma atividade.

Esse processo de comunicação do objeto é feito através de um processo denominado troca de mensagens. Assim, cada mensagem acaba sendo requisição para que um objeto execute uma operação específica.

Os códigos da classe base e das classes estendidas são os códigos desenvolvidos no capítulo anterior. Foi feita essa opção, pois as classes “Animal”, “Cachorro” e “Cobra” têm implementados todos os conceitos explicados.

Para exemplificar a troca de mensagens entre objetos, será desenvolvida a classe “Principal.java”. A figura a seguir demonstra a estrutura desses arquivos no Windows Explorer.



Fonte: o autor (2021).

O código a seguir implementa uma instância da classe “Cachorro” em um objeto chamado “cachorro”. Observe que no construtor da classe são definidos os atributos do objeto. Depois disso, é impressa na saída padrão a cor do cachorro, que foi definida no construtor e, por fim, é emitido o som do cachorro.

```java
public class Principal {
    public static void main(String args[]){
        //Instanciando o objeto cachorro da classe Cachorro
        Cachorro cachorro = new Cachorro("Amarelo", 5, true, "Vira lata", false);

        //Imprimindo o atributo cor
        System.out.println("A cor do cachorro é: " + cachorro.getCor() + "\n");

        //Imprimindo o som do cachorro
        System.out.println("O som do cachorro é:");
        cachorro.emitir_som();
    }
}
```

Fonte: o autor (2021).

A saída do código anterior é apresentada adiante:

```
A cor do cachorro é: Amarelo

O som do cachorro é:
Auau
```

Fonte: o autor (2021).

Para fixar o conteúdo, o código adiante implementa uma instância da classe “Cobra” em um objeto chamado “cobra”. Observe que no construtor da classe são definidos os atributos do objeto. Depois disso, é impressa na saída padrão a cor da cobra, que foi definida no construtor e modificada através do método “setCor”, e por fim é emitido o som da cobra.

```java
public class Principal {
    public static void main(String args[]){
        //Instanciando o objeto cobra da classe Cobra
        Cobra cobra = new Cobra("Marrom", 0.5, true, "Cobra cipó", true);

        //Alterando o atributo da cor da cobra
        cobra.setCor("Preta");

        //Imprimindo o atributo cor do objeto cobra
        System.out.println("A cor da cobra é: " + cobra.getCor() + "\n");

        //Imprimindo o som da cobra
        System.out.println("O som da cobra é:");
        cobra.emitir_som();
    }
}

```

Fonte: o autor (2021).

A saída do código anterior é apresentada a seguir:

```
A cor da cobra é: Preta

O som da cobra é:
shi
```

Fonte: o autor (2021).

4.6 Síntese da Unidade

Nessa Unidade, vimos a aplicação dos conceitos de construtores, encapsulamento, polimorfismo e de comunicação com a troca de mensagem com a linguagem de programação Java.

4.7 Para saber mais

Recomendamos que seja feita a leitura do capítulo 6 do livro “Java: como programar”, dos autores Deitel e Deitel, da editora Prentice-Hall, para se aprofundar um pouco mais no conteúdo abordado.

4.8 Aprendendo

Diversos objetos do mundo real podem ser abstraídos e enviados para o mundo da orientação a objetos. Portanto, imagine alguns objetos que poderiam ser representados de acordo com o paradigma de orientação a objetos.

4.9 Praticando

Imagine objetos que possam representar os conceitos aplicados nessa Unidade.
Posteriormente, desenvolva classes na linguagem de programação Java utilizando os conceitos de encapsulamento, construtor, herança e polimorfismo. Depois que as classes estiverem devidamente implementadas, desenvolva uma classe que realize a troca de mensagens entre as classes.

4.10 Referências

DEITEL, H. M.; DEITEL, P. J. **Java:** Como Programar. Rio de Janeiro: Prentice-Hell, 2005.

UNITAU digital

ISBN: 978-65-86914-53-5