Fundamentos da Programação Orientada a Objetos

UNIDADE 03

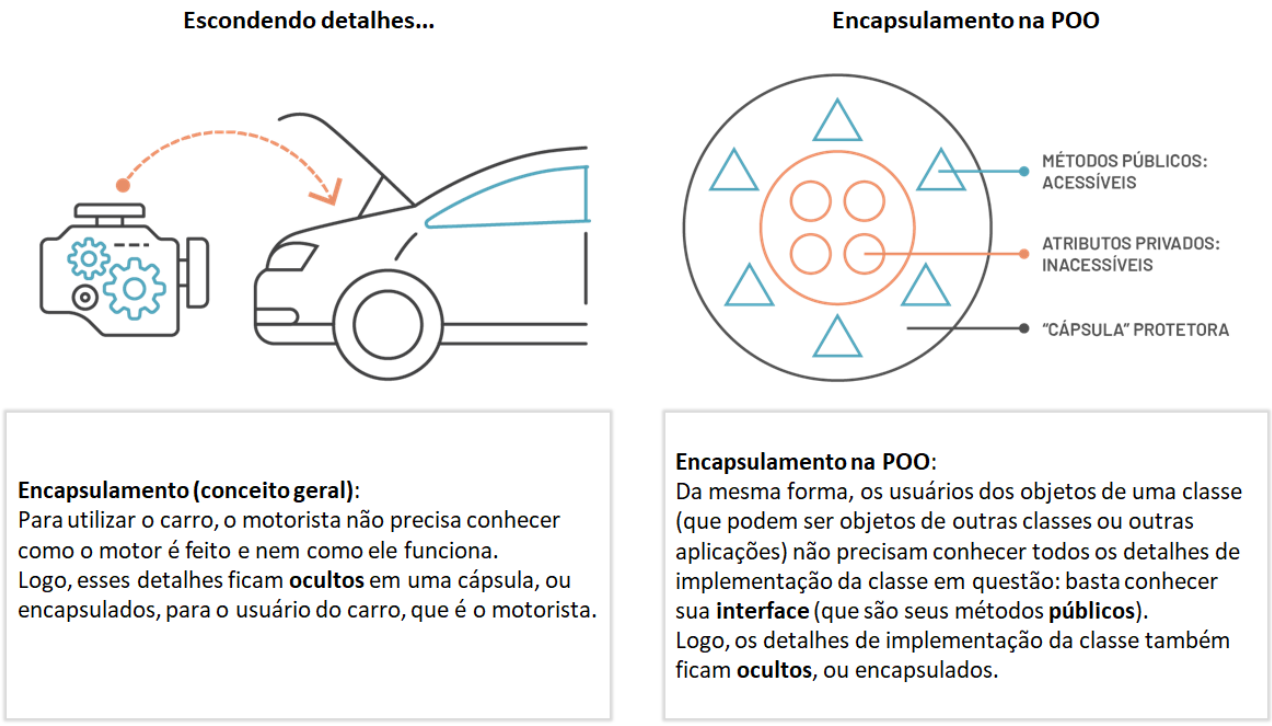
Encapsulamento

*Olá! Nesta Unidade, criaremos programas que ocultam os detalhes de implementação de uma classe, impedindo o acesso direto ao **estado** do objeto (**atributos**). Nesse caso, o acesso aos atributos ocultos pode ser feito apenas por meio de sua **interface** (**métodos públicos**), como ilustrado na Figura 1 a seguir.*

Vamos recordar: **abstrair** é observar um todo, com o objetivo de **isolar um elemento, ou um conceito, com o qual precisamos trabalhar**, excluindo os demais.

Usamos um processo de **abstração** para realizar o **encapsulamento** das classes, que consiste em separar a visão do **criador dos objetos** da visão do **usuário desses objetos**. Para isso, o criador do objeto, ou programador, "esconde" a **parte interna da classe** (detalhes de lógica e atributos) da parte **externa da classe**, que é acessada pelos **usuários** dos objetos da classe. A visão externa da classe é chamada de **interface da classe.**

Figura 1 – Encapsulamento



Fonte: Autores (2021).

Modificador de acesso: público, privado e pacote

O Java utiliza algumas palavras-chave para ocultar os detalhes de implementação das classes, que são os **modificadores de acesso**. Eles são utilizados para especificar quais membros da classe (atributos ou métodos) podem ser visíveis ou acessíveis por outras classes.

> O **conjunto de métodos** de uma classe especificados como **acessíveis por outras classes** constitui a **interface** dessa classe.

Na Figura 2 estão apresentados os modificadores de acesso **público**, **privado** e **protegido**. O modificador protegido será trabalhado novamente quando estivermos estudando a **Unidade 5 – Hierarquia**.

Modificador **público** em Java = ***`public`***:

Indica que a **classe**, **atributo** ou **método** é visível por qualquer outra classe, em qualquer pacote.

Modificador ***default*** (padrão) em Java = não usa palavra-chave; vazio:

Indica que a **classe**, **atributo** ou **método** é apenas por classes que estejam no mesmo pacote (***package***) da classe em que foram declarados.

Modificador **protegido** em Java = ***protected****:*

Indica que o **atributo** ou **método** é visível apenas por classes que estejam no mesmo pacote (***package***) da classe em que foram declarados, ou em **classes filhas** (subclasse) de qualquer pacote*.

Modificador **privado** em Java = ***private****:*

Indica que o atributo ou método é visível apenas dentro da própria classe em que foi declarado.

Figura 2 – Visibilidade x modificadores de acesso

Visibilidade				
Modificador em Java	***Class***	***Package***	***Subclasse****	**Mundo**
*Sem modificador (*default*)*	☑	☑	☒	☒
`public`	☑	☑	☑	☑
`protected` *	☑	☑	☑	☒
`private`	☑	☒	☒	☒

*Veremos detalhes do conceito de **classes filha** (subclasses) com o modificador **protected** na Unidade 5 – Hierarquia.

Fonte: Autores (2021).

Para exemplificar, vamos pensar em uma solução computacional que é concebida a partir de um **exercício de abstração**, em que precisamos identificar os elementos que interessam a um determinado *software*.

Membros públicos

Como vimos, um membro (atributo ou método) **público** é **visível e acessível por qualquer outra classe**, logo:

- Um **método público** pode ser **chamado** de qualquer classe.
- Um **atributo público** pode ser **lido** ou **escrito** a partir de qualquer classe.

Figura 3 – Exemplo de atributos e métodos públicos

```
</>
```

```
1  class A {
2     public static void g( ) // método A.g() é público
3     { … }
4     public static int k;     // atributo A.k é público
5  }
```

```
</>
```

```
1  class B {
2     static void f( ) // método B.f() é default
3     {
4        A.g();        // B.f() chama o método A.g()
5        int x = A.k;  // B.f() lê o atributo A.k
6        A.k = 10;     // B.f() escreve ou modifica o atributo
A.k
7     }
8  }
```

Fonte: Autores (2021).

Membros privados

Como vimos, um membro (atributo ou método) **privado** é **visível e acessível apenas pela própria classe**, logo:

- Um **método privado** pode ser **chamado apenas dentro na própria classe**, em que foi declarado.
- Um **atributo privado** pode ser **lido** ou **escrito apenas dentro na própria classe**, em que foi declarado.

Figura 4 – Exemplo de **atributos** e **métodos privados**

```
</>
```

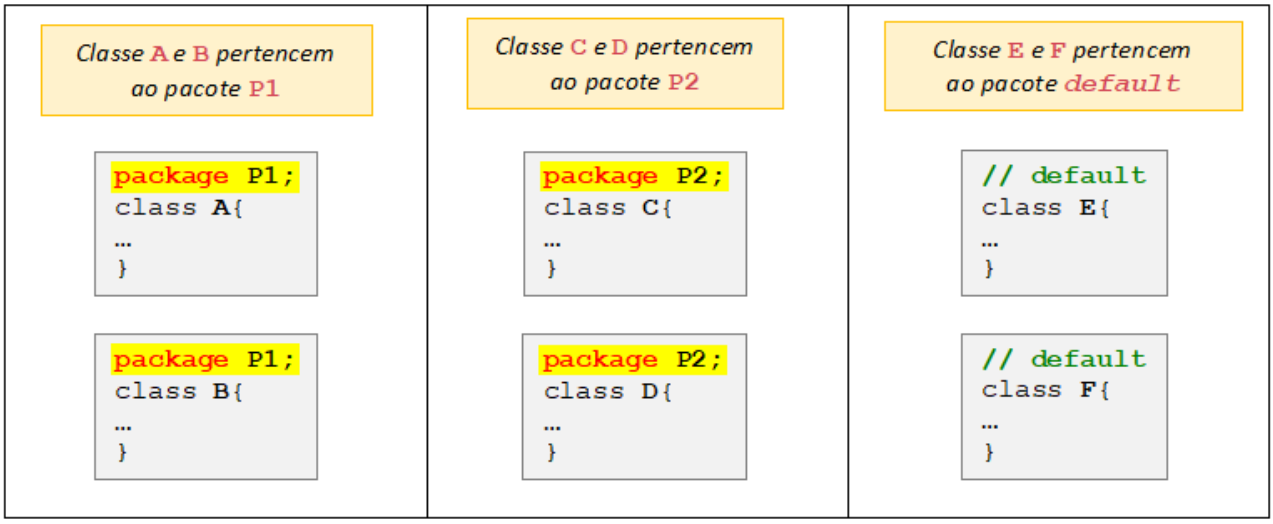
```
 1  class A {
 2     private static void g( ) // método A.g() é privado
 3     { … }
 4     private static int k;     // atributo A.k é privado
 5
 6    static void f( )          // método A.f() é default
 7     {
 8        g();       // método A.g() é chamado na própria
classe, em A.f()
 9        int x = k;
10        k = 10;    // atributo A.k é lido na própria classe,
em A.f())
11     }
12  }
```

Fonte: Autores (2021).

Membros de pacote

- Pacotes (***packages***) são usados para **agrupar classes**.
- Um **membro** (atributo ou método) de uma **classe declarada dentro de pacote** é **visível e acessível** por todas as **classes desse mesmo pacote** e somente por essas.
- **Importante:** quando o **modificador não é especificado**, a classe pertence ao pacote ***default*** (padrão).

Figura 5 – Exemplo de **atributos** e **métodos** no **mesmo pacote**

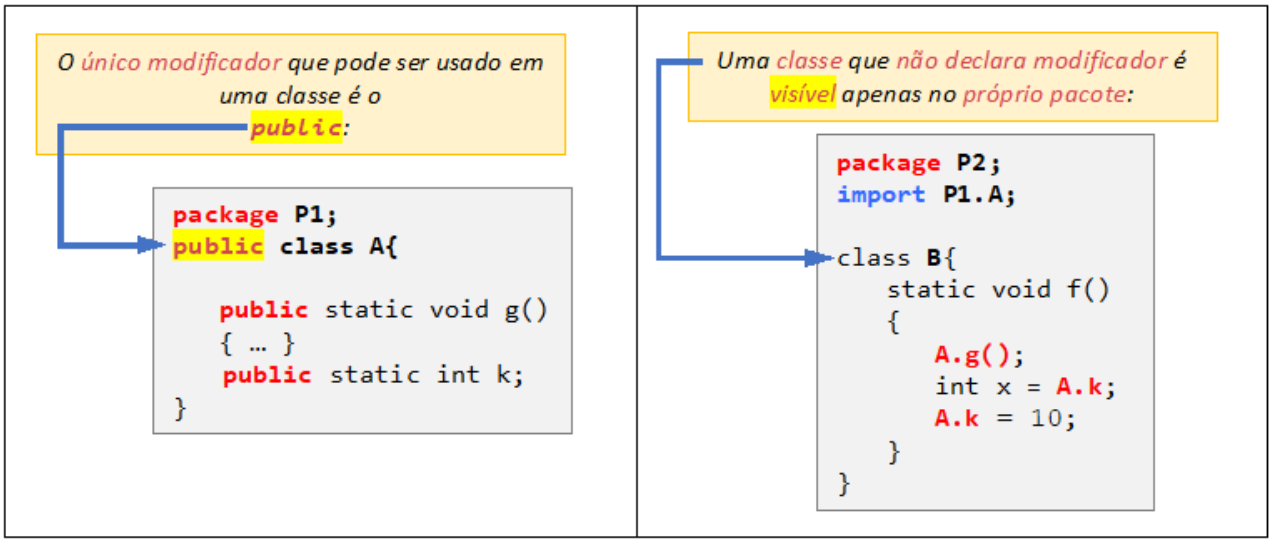


Fonte: Autores (2021).

Classes públicas

Uma classe pública é visível e acessível a partir de **classes de outros pacotes**.

Figura 6 – Exemplo de **classes públicas**



Fonte: Autores (2021).

EXERCÍCIO

Encapsulamento

Verifique a sua compreensão em relação à criação e utilização de objetos que utilizam encapsulamento – detalhes apresentados na Figura 7 a seguir.

1. Crie um projeto com as classes **A** e **B** no pacote **P1**, conforme indicado.
2. Observe que n**ão são declarados modificadores de acesso**. Logo, o modificador é ***default*** (padrão vazio).
3. Observe que o atributo **A.k** não tem valor declarado e, nesse caso, seu valor *default* é zero.
4. Observe que o método **A.g()** apenas apresenta o valor de **A.k** na tela.
5. Observe que a classe **B** tem acesso aos membros da classe **A**, pois o modificador dos seus membros é o ***default*** e ambas as classes estão no mesmo pacote **P1**.
6. Observe que o método **B.f()** consegue alterar o atributo **A.k** – tem permissão de acesso, pois o modificador de **A.k** é o ***default***.

7. Verifique as questões a seguir no seu ambiente de desenvolvimento (IDE) e responda:

    a. É possível usar o modificador ***private*** antes da palavra-chave ***class***, tanto na classe **A** quanto na classe **B**? O que acontece?
    b. É possível usar o modificador ***private*** nos métodos **A.g ()** e **B.f ()**? Justifique.
    c. É possível usar o modificador ***protected*** nos métodos **A.g ()** e **B.f ()**? Justifique.

Figura 7 – Exercício encapsulamento: classe **A** e classe **B**

</>

```
 1  package P1;
 2
 3  class A{
 4      static int k; // valor
default = zero
 5
 6
 7      static void g() {
 8
System.out.println("------------
------");
 9
System.out.println("... Método
A.g (sem modificador)");
10
System.out.println("... Valor de
A.k = " + k);
11
System.out.println("------------
------");
12      }
13  }
```

```
Console
Método f (sem modificador)
-----------------
... Método A.g (sem modificador)
... Valor de A.k = 0
-----------------
Valor de x = 0
Valor de k = 10
```

</>

```
 1  package P1;
 2
 3  class B {
 4      static void f() {
 5          System.out.println("Método f (sem
modificador)");
 6          A.g();
 7          int x = A.k;
 8          System.out.println("Valor de x = "
+ x);
 9          A.k = 10;
10          System.out.println("Valor de k = "
+ A.k);
11      }
12
13      public  static void main (String
args[]) {
14          f();
15      }
16  }
```

Fonte: Autores (2021).

Resolução do exercício

a. Não é possível. Gera erro de compilação, pois a classe precisa ser pública ou default (vazio).
b. O método A.g () não pode ter modificador private porque ele é utilizado nos objetos da classe B . O método B.f () poderia ter modificador private porque apenas é utilizado dentro da própria classe B.
c. Poderíamos utilizar o modificador protected em ambos os métodos A.g () e B.f (), pois são classes que estão no mesmo pacote e, com esse modificador, os métodos estariam acessíveis entre as classes.

Método construtor

Vamos recordar o que já apresentamos na **Unidade 2 – Classes e Objetos**, sobre o **método construtor.** Em Java, esse é um método **especial**, usado para inicializar objetos. O construtor é chamado quando um objeto de uma classe é criado ou instanciado. Ele pode ser usado para definir valores iniciais para atributos de objeto, conforme exemplificado na Figura 8 a seguir.

1. Crie um projeto com a classe **carro**, conforme indicado.

2. Observe que **carro** tem um atributo privado: **marca** (*string*).

3. Observe o método **construtor**:

   a. Tem o mesmo nome da classe **carro**.
   b. É **público**.
   c. Não declara tipo de retorno: **o retorno é o próprio objeto** de **carro**.
   d. Sua chamada (ou invocação) é feita com a palavra-chave ***new***.
   e. É usado para iniciar o atributo **marca**.

4. Execute o programa e verifique o resultado.

Figura 8 – Exercício com método construtor

</>

```
 1  public class Carro {
 2      private String  marca;  // atributo
oculto (encapsulamento / privado)
 3
 4      public Carro(String marca) { //
Método construtor
 5          this.marca = marca;
 6      }
 7      public void imprimir() { // Método
que exibe o valor do atributo na tela
 8          System.out.println("Marca:  " +
this.marca);
 9      }
10      public static void main(String[]
args) {
11          Carro meuCarro = new
Carro("Ford"); // instancia Carro
12          meuCarro.imprimir();    //
método público para exibir dados
13                                          //
privados / encapsulados de carro
14      }
15  }
```

```
Console
Marca:  Ford
```

Fonte: Autores (2021).

Métodos *getter* e *setter*

Quando o encapsulamento oculta atributos da classe, usando o modificador de acesso ***private***, é possível permitir o acesso a esses atributos de uma maneira controlada. A prática mais comum para fazer isso é criar dois métodos padrão ***getter*** (que **obtém** ou **retorna** alguma coisa) e ***setter*** (que **atribui** ou **modifica** alguma coisa). Esses métodos compõem a interface da classe e visam a prover um acesso controlado a atributos encapsulados. Veja como são declarados:

1. Padrão do método ***getter*** – retorna o valor do atributo: `getNomeAtributo`.
2. Padrão do método ***setter*** – muda o valor do atributo: `setNomeAtributo`.
3. Padrão do método ***is*** – retorna o valor boolean do atributo: `isAtributo` .

Verifique e execute os exemplos de ***getter*** e ***setter*** nas Figuras 9 e 10 a seguir.

1. Crie um projeto com a classe **carro** modificada, conforme indicado.

2. Observe que **carro** tem dois atributos privados: **marca** (*string*) e **ligado** (*boolean*).

3. Observe o padrão dos métodos ***getter*** e ***setter*** para esses atributos – esses padrões para nome de método são boas práticas de programação:

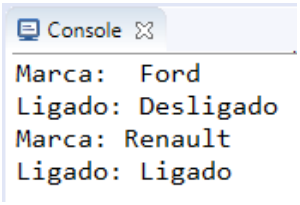
a. *public String get***Marca**() – retorna valor do atributo privado "marca";
b. *public void set***Marca**(*String* marca) – altera valor do atributo privado "marca";
c. *public boolean is***Ligado**() – retorna valor do atributo privado "ligado";
d. *public void set***Ligado**(*boolean* ligado) – altera valor do atributo privado "ligado";

4. Observe o **operador ternário** (veja novamente a **Unidade 1 – Dados e Variáveis**):

a. Segue o padrão **(expressão booleana) ? código 1 : código 2;**
b. Como no comando: `meuCarro.isLigado()==true?"Ligado":"Desligado"`
c. Assim, `meuCarro.isLigado() pode retornar true, que imprime "Ligado".`
d. *Ou pode retornar* `false`*, que imprime* `"Desligado".`

Figura 9 – Exercício com métodos ***getter*** *e* ***setter***

```
 1  public class Carro {
 2      private String  marca;  // atributo
oculto (encapsulado / privado)
 3      private boolean ligado; // atributo
oculto (encapsulado / privado)
 4
 5      public Carro(String marca, boolean
ligado) { // Método construtor
 6          this.marca = marca;
 7          this.ligado = ligado;
 8      }
 9      public String getMarca() { //
Retorna valor do atributo privado “marca”;
10          return marca;
11      }
12      public void setMarca(String marca) {
// Altera valor de “marca”;
13          this.marca = marca;
14      }
15      public boolean isLigado() {  //
Retorna valor do atributo privado “ligado”;
16          return ligado;
17      }
18      public void setLigado(boolean
ligado) { //  Altera valor de “ligado”;
19          this.ligado = ligado;
20      }
21
22      public static void main(String[]
args) {
23          Carro meuCarro = new
Carro("Ford",false); //instancia Carro
24          System.out.println("Marca:  " +
meuCarro.getMarca()); //Retorna valor
25          System.out.println("Ligado: " +
26
(meuCarro.isLigado()==true?"Ligado":"Desliga
do"));
27
28          meuCarro.setMarca("Renault");
//Altera valor de atributo privado
29          meuCarro.setLigado(true);
//Altera valor de atributo privado
30
31          System.out.println("Marca: " +
meuCarro.getMarca()); // Retorna valor
32          System.out.println("Ligado: " +
33
```



```
(meuCarro.isLigado()==true?"Ligado":"Desliga
do"));
 34      }
 35  }
```

Fonte: Autores (2021).

EXERCÍCIO

Interface criada com métodos *getter* e *setter*

Verifique o acesso controlado ao estado dos objetos de uma classe, conforme indicado na Figura 10.

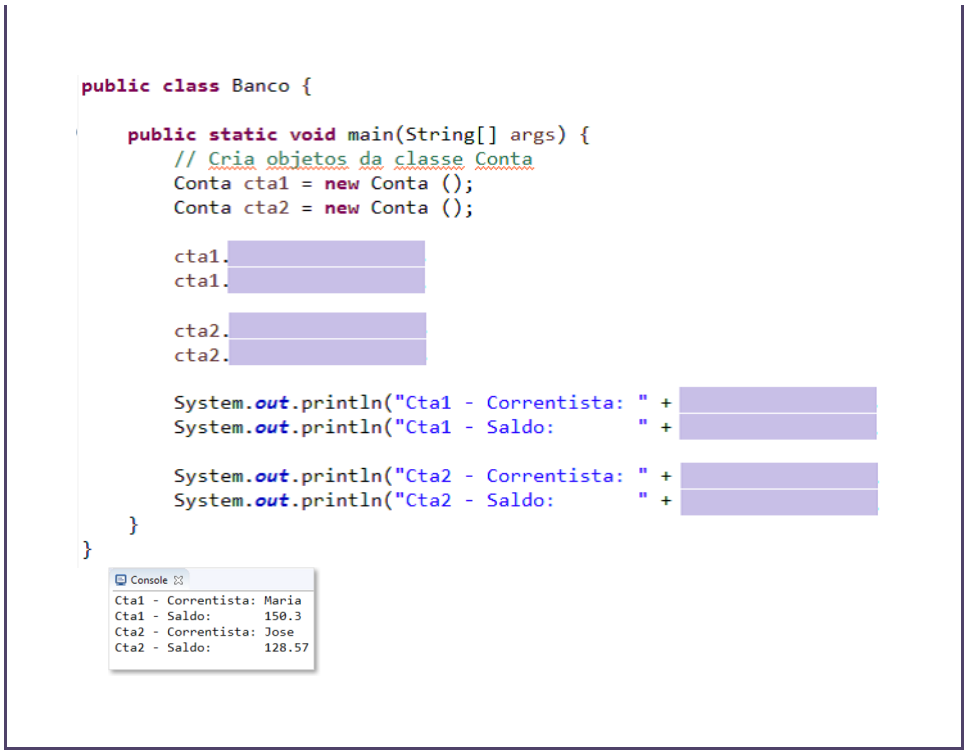
1. Crie um projeto com as classes **banco** e **conta**, como indicado.
2. Complete o código da classe **banco** com métodos padrão ***getter*** e ***setter*** para obter a saída no console, conforme indicado.

Figura 10 – Interface com métodos ***getter*** e ***setter***

</>

```
 1  public class Conta {
 2     private double saldo;
 3     private String dono;
 4
 5     public String getDono() {
 6        return dono;
 7     }
 8     public void setDono(String dono) {
 9        this.dono = dono;
10     }
11     public double getSaldo() {
12        return saldo;
13     }
14     public void setSaldo(double saldo) {
15        this.saldo = saldo;
16     }
```

</>



```
public class Banco {

    public static void main(String[] args) {
        // Cria objetos da classe Conta
        Conta cta1 = new Conta ();
        Conta cta2 = new Conta ();

        cta1.
        cta1.

        cta2.
        cta2.

        System.out.println("Cta1 - Correntista: " +
        System.out.println("Cta1 - Saldo:       " +

        System.out.println("Cta2 - Correntista: " +
        System.out.println("Cta2 - Saldo:       " +
    }
}
```

```
Console
Cta1 - Correntista: Maria
Cta1 - Saldo:       150.3
Cta2 - Correntista: Jose
Cta2 - Saldo:       128.57
```

Fonte: Autores (2021).

EXERCÍCIO

Fixação do conceito de encapsulamento

1. Crie um projeto com as classes **arquiteto** e **circulo**, no mesmo pacote default, como indicado.
2. Execute o projeto a partir da classe **Arquiteto**.
3. Observe o uso dos modificadores ***public*** e ***private***.
4. Observe as diferentes formas de instanciar a classe **circulo**.
5. Remova os modificadores ***public*** dos membros da classe **circulo** e repita o teste. Por que ainda funciona?
6. Crie um projeto com **dois pacotes**, denominados **especialista** e **matemática**, contendo, respectivamente, as classes **arquiteto** e **circulo**.

    a. Será necessário incluir os comandos ***package*** e ***import***.
    b. Quais métodos e/ou atributos de que classe precisam mudar para visibilidade ***public***?

Figura 11 – Exercício fixação de encapsulamento: classes **arquiteto** e **circulo**

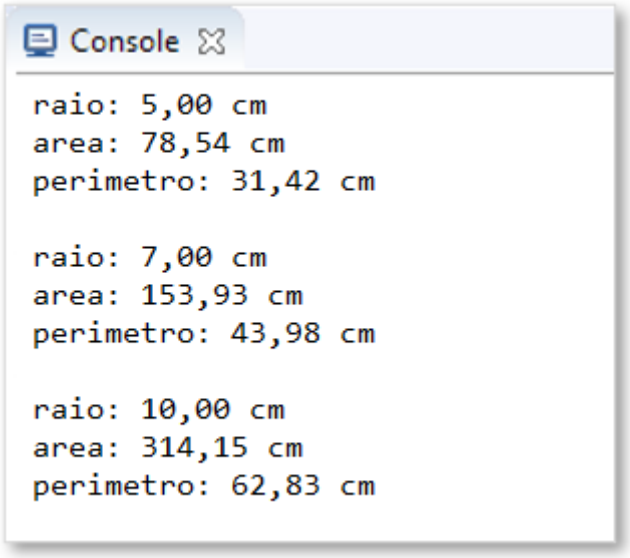
</>

```
 1  public class Arquiteto {
 2      private String nome;
 3      private int idade;
 4
 5      // Método construtor (tem o mesmo nome
da classe)
 6      public Arquiteto(String nome, int
idade) {
 7      this.nome = nome;   // "this" =
referência ao atual objeto
 8      this.idade = idade;
 9      }
10      public void exibirDadosPessoais(){
11          System.out.println( nome);
12          System.out.println( idade+ "
anos");
13      }
14      public void trabalhe (double r1,
double r2, double r3) {
15          Circulo a= new Circulo(r1); //
define e instancia o objeto "a"
16          double x = a.area();
17          double y = a.perimetro();
18          imprima(r1,x,y);
19
20          Circulo b= new Circulo(r2);
//define e instancia o objeto "b"
21          x = b.area();
22          y = b.perimetro();
23          imprima(r2,x,y);
24
25          b = new Circulo(r3);        //
instancia novamente o objeto "b"
26          x = b.area();
27          y = b.perimetro();
28          imprima(r3,x,y);
29      }
30
31      private void imprima(double raio,
double area, double perimetro) {
32          // %.2f = imprime float com 2
casas decimais
33          // \n   = salto de linha
34          System.out.printf("raio: %.2f
cm\n", raio);
35          System.out.printf("area: %.2f
cm\n", area);
```

```
 36          System.out.printf("perimetro: %.2f
cm\n", perimetro);
 37          System.out.println();
 38      }
 39
 40      public static void main (String[]
args) {
 41          Arquiteto a1 = new Arquiteto
("Oscar Niemeyer", 104);
 42          a1.trabalhe(5, 7, 10);
 43      }
 44  }
```

</>

```
  1  public class Circulo {
  2      private static double PI
= 3.141516;
  3      private double raio;
  4
  5      public Circulo(double
raio) {
  6          this.raio = raio;
  7      }
  8      public double area(){
  9          return PI * raio *
raio ;
 10      }
 11      public double perimetro
(){
 12          return 2 * PI *
raio;
 13      }
 14  }
```



Fonte: Autores (2021).

Resolução do exercício

5. Porque um atributo ou método default (identificados pela **ausência** de modificador) pode ser acessado por objetos de todas as classes que estiverem no mesmo pacote que a classe que possui o atributo ou o método. Nesse exemplo, arquiteto e circulo estão no mesmo pacote.

6.Respostas

a. Classe arquiteto:

```
package Especialista;

import Matematica.Circulo;
```

Classe circulo:

```
package Matematica;
```

b. Métodos da classe **círculo**, que está no package Matematica, devem todos ser **públicos**, para serem acessados pelo objeto de Arquiteto, que está no package Especialista.

Classe *string*

As ***strings***, muito utilizadas na programação Java, são uma **sequência de caracteres**. Todos os tipos de dados utilizados no Java, com exceção dos tipos primitivos (***int***, ***double***, ***char*** e ***boolean***), são objetos. Logo, uma ***string***, contida na biblioteca padrão **java.lang**, também é um objeto e deve ser **declarada** e **instanciada**.

Pratique a criação e manipulação de ***strings*** nos exercícios a seguir.

EXERCÍCIO

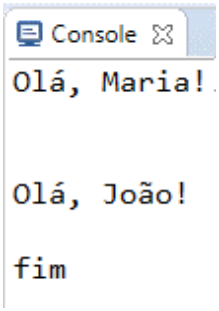
Criação de *strings*

1. Crie um projeto com a classe ***StringDemo***, como indicado.
2. Observe as diferentes formas de criar uma ***string***.
3. Execute o projeto e verifique o resultado.

Figura 12 – Exemplos de criação de ***strings***

</>

```
 1  public class StringDemo {
 2
 3      public static void
main(String[] args) {
 4          String ola = "Olá";
// Declara e instancia
diretamente
 5
 6          String nome1 = new
String("Maria"); // Instancia
com new
 7          String nome2 = new
String("João");  // Instancia
com new
 8
 9          // Declara e
instancia com a concatenação de
strings
10          String saudacao =
ola + ", " + nome1 + "!\n\n";
11
System.out.println(saudacao);
12
13          // Altera o valor de
saudacao pela atribuição do
valor de ola
14          // Não é preciso
declarar saudacao novamente
15          saudacao = ola;
16
17          // Altera o calor de
saudacao, que recebe
18          // o resultado da
concatenacao de mais strings
19          // É uma alternativa
que utiliza o método
20                //
String.concat(String s):
21          saudacao =
saudacao.concat(", " + nome2);
22          saudacao =
saudacao.concat("!\n\n");
23
24          //Mais uma
alternativa de concatenação
25
System.out.println(saudacao +
"fim");
26      }
27  }
```



Fonte: Autores (2021).

EXERCÍCIO

Utilizando métodos da classe *string*

1. Crie um projeto com a classe ***StringExemples***, como indicado.
2. Observe os exemplos de utilização dos métodos da classe ***string***.
3. Execute o projeto e verifique o resultado.

Figura 13 – Exemplos de utilização de métodos da classe *string*

</>

```
 1  public class StringExemples
{
 2
 3     public static void
main(String[] args) {
 4        String txt;
 5
 6        txt =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 7
System.out.println("Comprimeto
de [" + txt + "] = " +
txt.length());
 8
 9        txt = "Olá Mundo!";
10
System.out.println(txt.toUpperCa
se());   // Saída = "OLÁ MUNDO!"
11
System.out.println(txt.toLowerCa
se());   // Saída = "olá mundo"
12
13        txt = "Localize a
posição da palavra 'eureka'
neste texto!";
14
System.out.println(txt.indexOf("
eureka")); // Saída = 31
15     }
16  }
17
```

```
Console
Comprimeto de [ABCDEFGHIJKLMNOPQRSTUVWXYZ] = 26
OLÁ MUNDO!
olá mundo!
31
```

Fonte: Autores (2021).

Métodos de *string*

A Figura 14 apresenta alguns dos métodos da classe ***string***. A lista completa, com exemplos de utilização, pode ser encontrada em ORACLE, 2021 e em W3SCHOOLS, 2021.

Figura 14 – Os métodos da classe **java.lang.String.**

Método	Descrição	Tipo de retorno
`charAt()`	Retorna o caractere no índice especificado (posição).	`char`
`concat(String str)`	Concatena a string especificada ao final desta string.	`string`
`equals(Object anObject)`	Compara esta string com o objeto especificado.	`boolean`
`equalsIgnoreCase(String anotherString)`	Compara esta string com outra string, ignorando diferenças de maiúsculas e minúsculas (case insensitive).	`boolean`
`indexOf(int ch)`	Retorna o índice dentro desta string da primeira ocorrência do caractere especificado.	`int`
`indexOf(String str)`	Retorna o índice dentro desta string da primeira ocorrência da substring especificada.	`int`
`isEmpty()`	Retorna verdadeiro se, e somente se, length () for 0.	`boolean`
`length()`	Retorna o comprimento desta string.	`int`
`replace(char oldChar, char newChar)`	Retorna uma nova string resultante da substituição de todas as ocorrências de oldChar nesta string por newChar.	`string`
`replace(char oldChar, char newChar)`	Retorna uma nova string resultante da substituição de todas as ocorrências de oldChar nesta string por newChar.	`string`
`substring(int beginIndex)`	Retorna uma nova string que é uma substring desta string.	`string`
`toLowerCase()`	Converte todos os caracteres desta string em minúsculas.	`string`
`toUpperCase()`	Converte todos os caracteres desta string em maiúsculas.	`string`
`valueOf(float f)`	Retorna a representação de string do argumento float.	`string`
`valueOf(int i)`	Retorna a representação de string do argumento int.	`string`

Fonte: Autores (2021).

Referências

GODOY, V. **Programação Orientada a Objetos I**. Curitiba: IESDE, 2019.

HORSTMANN, C. S.; CORNELL, G. **Core Java – Volume I**. 8. ed. São Paulo: Pearson, 2010.

ORACLE, 2021. **Class String**. Disponível em: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html. Acesso em: 20 jan. 2021.

SCHILDT, H. **Java para Iniciantes**. Porto Alegre: Bookman, 2015.

W3SCHOOLS, 2021. **Java String Methods**. Disponível em: https://www.w3schools.com/java/java_ref_string.asp. Acesso em: 20 jan. 2021.