

Programação Orientada a Objetos

Encapsulamento, atributos públicos e privados.

Encapsulamento, atributos públicos e privados.

Encapsulamento é uma forma de fazer com que os objetos mantenham suas informações de estado privadas, tornando seu comportamento oculto para o mundo externo.

Com encapsulamento não é possível alterar o estado interno dos objetos atuando diretamente neles.

Para manipular as informações dos objetos, enviamos “mensagens” aos mesmos, utilizando funções como *get* e *set*.

Encapsulamento, atributos públicos e privados.

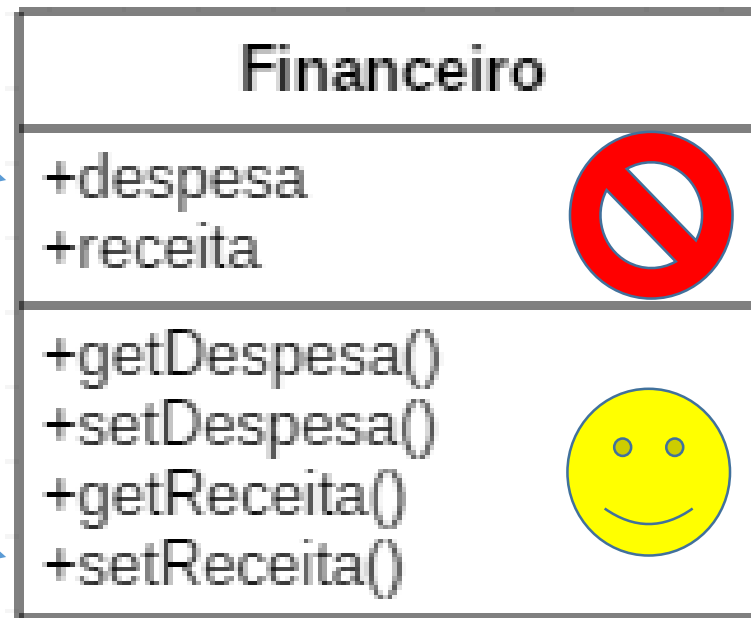
Nas linguagens Java e C++, por exemplo, é possível definir propriedades como públicas ou privadas. As propriedades privadas não podem ser acessadas diretamente, necessitando da implementação dos métodos *get* e *set*.

No Python o conceito de encapsulamento (ocultar dados e métodos) não é implícito, pois não existem palavras reservadas como no Java ou C++ (*public*, *private* e *protected*).

Encapsulamento, atributos públicos e privados.

No Java, quando definimos um atributo como privado (private), ele não é acessível fora da classe.

Sendo assim, são criados métodos get e set para manipular estes atributos.



Encapsulamento, atributos públicos e privados.

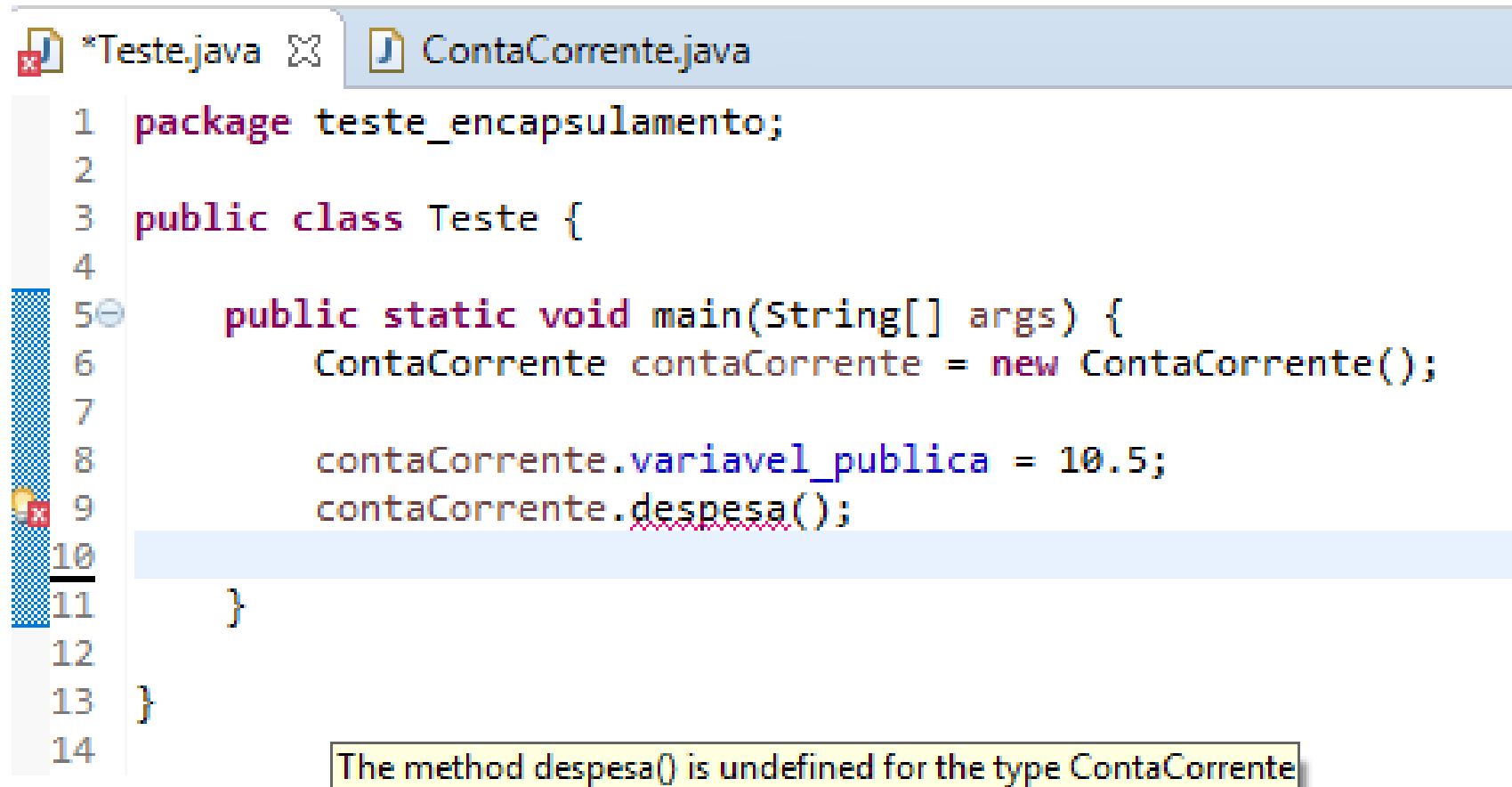
```
Teste.java  ContaCorrente.java ✕
1 package teste_encapsulamento;
2
3 public class ContaCorrente {
4     private Double despesa;
5     private Double receita;
6     public Double variavel_publica;
7
8     public Double getDespesa() {
9         return despesa;
10    }
11
12    public void setDespesa(Double despesa) {
13        this.despesa = despesa;
14    }
15
16    public Double getReceita() {
17        return receita;
18    }
19
20    public void setReceita(Double receita) {
21        this.receita = receita;
22    }
23
24 }
```

```
*Teste.java ✕  ContaCorrente.java
1 package teste_encapsulamento;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         ContaCorrente contaCorrente = new ContaCorrente();
7
8         contaCorrente.
9
10    }
11
12 }
13
14
```

- variavel_publica : Double - ContaCorrente
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- getDespesa() : Double - ContaCorrente
- getReceita() : Double - ContaCorrente
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- setDespesa(Double despesa) : void - ContaCorrente
- setReceita(Double receita) : void - ContaCorrente
- toString() : String - Object
- wait() : void - Object

Press 'Ctrl+Space' to show Template Proposals

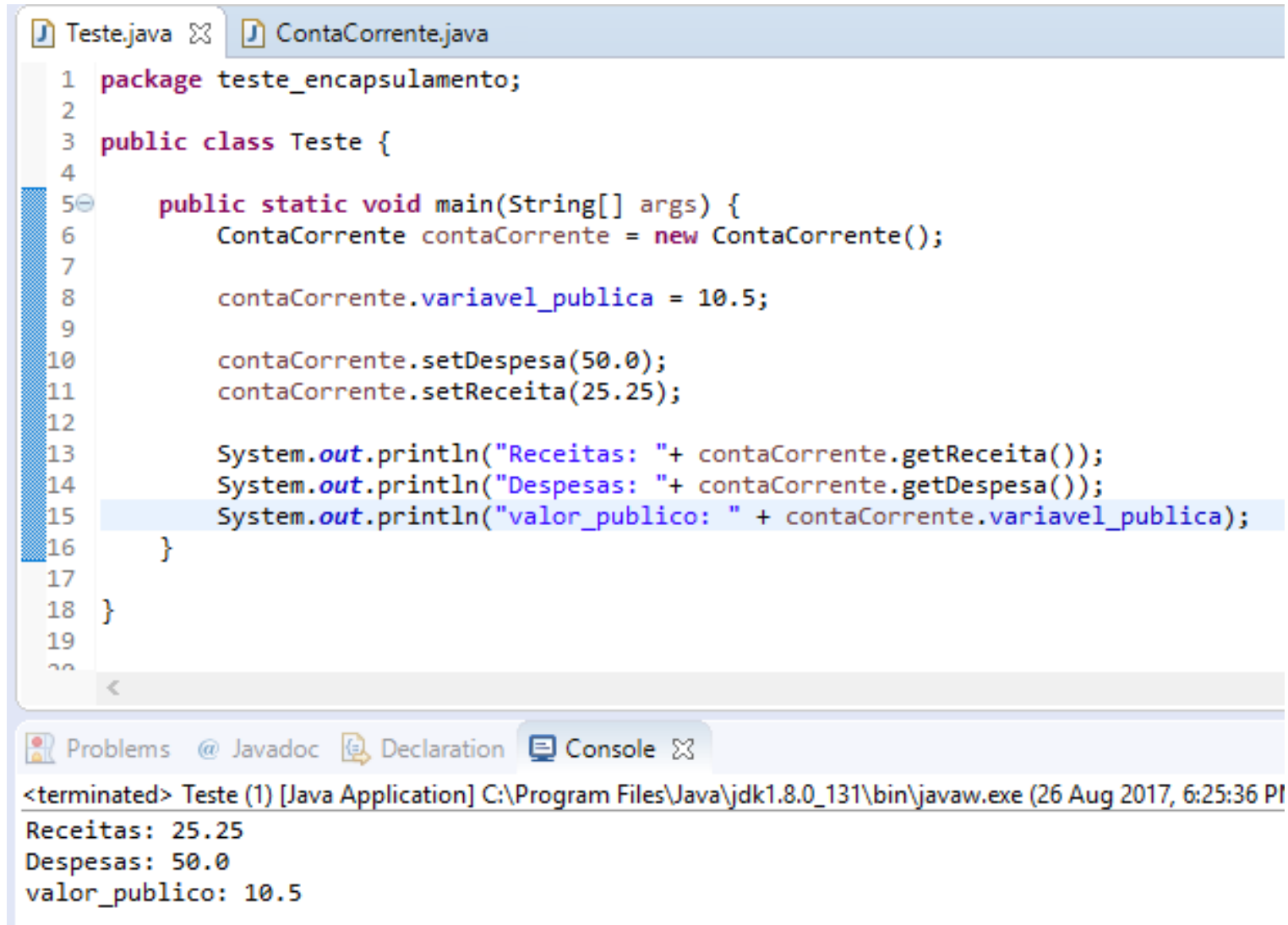
Encapsulamento, atributos públicos e privados.



```
1 package teste_encapsulamento;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         ContaCorrente contaCorrente = new ContaCorrente();
7
8         contaCorrente.variavel_publica = 10.5;
9         contaCorrente.despesa();
10
11     }
12
13 }
14
```

The method despesa() is undefined for the type ContaCorrente

Encapsulamento, atributos públicos e privados.



The screenshot displays an IDE with two tabs: `Teste.java` and `ContaCorrente.java`. The `Teste.java` tab is active, showing the following code:

```
1 package teste_encapsulamento;
2
3 public class Teste {
4
5     public static void main(String[] args) {
6         ContaCorrente contaCorrente = new ContaCorrente();
7
8         contaCorrente.variavel_publica = 10.5;
9
10        contaCorrente.setDespesa(50.0);
11        contaCorrente.setReceita(25.25);
12
13        System.out.println("Receitas: " + contaCorrente.getReceita());
14        System.out.println("Despesas: " + contaCorrente.getDespesa());
15        System.out.println("valor_publico: " + contaCorrente.variavel_publica);
16    }
17
18 }
19
20
```

Below the code editor, the `Console` tab is selected, showing the output of the program:

```
<terminated> Teste (1) [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (26 Aug 2017, 6:25:36 PM)
Receitas: 25.25
Despesas: 50.0
valor_publico: 10.5
```

Encapsulamento, atributos públicos e privados.

Variáveis de instância privadas, que são acessadas somente do interior do objeto não existem no Python.

Porém, existe uma convenção seguida pelos programadores Python: Um nome prefixado com um underscore (por exemplo `_carro`) deve ser tratado como não público. Seja uma função, método ou membro de dados.

Encapsulamento, atributos públicos e privados.

```
class Carro(object):  
  
    def __init__(self, marca, modelo):  
        self._marca = marca  
        self._modelo = modelo  
  
    def get_modelo(self):  
        return self._modelo  
  
    def set_modelo(self, modelo):  
        self._modelo = modelo  
  
carro = Carro("Ford", "Ranger")  
print(carro._marca)  
carro._marca = "Fiat"  
print(carro._marca)  
carro.set_modelo("Uno")  
print(carro.get_modelo())  
print(carro.get_modelo())
```

Apesar de ser possível acessar o “_marca” diretamente, seu nome indica que é um método “não público”, sendo assim, é melhor não usá-lo diretamente.

Encapsulamento, atributos públicos e privados.

Para criar um atributo “privado” no Python, temos que nomear o atributo iniciando com dois underscores. Exemplo:

Quando definimos com dois underscores, o atributo não é

Acessível externamente pelo

Seu nome “__despesa”.

O Python substitui seu nome por

`_nomedaclassa__variável`: `_Financeiro__despesa`.

Na verdade, os atributos não se tornam realmente privados, porque mesmo assim, ainda é possível acessá-los externamente, mesmo que com outro nome.

Financeiro
+ __despesa + __receita
+get_despesa() +set_despesa() +get_receita() +set_receita()

Encapsulamento, atributos públicos e privados.

```
class Carro(object):  
    marca = "Ford"  
    __modelo = "Focus"
```

```
carro = Carro()  
print(dir(carro))
```

dir retorna a lista de atributos do objeto informado.

```
['_Carro__modelo', '__class__', '__delattr__',  
 '__dict__', '__dir__', '__doc__', '__eq__',  
 '__format__', '__ge__', '__getattr__', '__gt__',  
 '__hash__', '__init__', '__init_subclass__', '__le__',  
 '__lt__', '__module__', '__ne__', '__new__',  
 '__reduce__', '__reduce_ex__', '__repr__',  
 '__setattr__', '__sizeof__', '__str__',  
 '__subclasshook__', '__weakref__', 'marca']
```

```
class Carro(object):  
    marca = "Ford"  
    __modelo = "Focus"  
  
carro = Carro()  
print(carro.)
```

marca	Carro
__setattr__(self, name, val...	object
__init__(self)	object
__annotations__	object
__class__	object
__delattr__(self, name)	object
__dict__	object
__doc__	object
__eq__(self, o)	object
__format__(self, format_spe...	object
__getattr__(self, name)	object

Did you know that Quick Definition View (Ctrl+Shift+I) works in completion lookups as well? >>

Encapsulamento, atributos públicos e privados.

```
class Carro(object):
    marca = "Ford"
    __modelo = "Focus"

    def get_modelo(self):
        return self.__modelo

    def set_modelo(self, modelo):
        self.__modelo = modelo

carro = Carro()
print(carro.marca)
carro.marca = "Fiat"
print(carro.marca)
carro.set_modelo("Uno")
print(carro.get_modelo())
carro.__Carro__modelo = "Palio"
print(carro.get_modelo())
```

Ford
Fiat
Uno
Palio

```
print(carro.__modelo)
AttributeError: 'Carro' object has no attribute '__modelo'
print(carro.modelo)
AttributeError: 'Carro' object has no attribute 'modelo'
```

Encapsulamento, atributos públicos e privados.

```
class Carro(object):  
    def __init__(self, fabricante):  
        self.__fabricante = fabricante  
  
    def set_fabricante(self, fabricante):  
        self.__fabricante = fabricante  
  
    def get_fabricante(self):  
        return self.__fabricante
```

```
carro = Carro("Toyota")  
print(dir(carro))
```

```
['_Carro__fabricante', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__',  
 '__eq__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__',  
 '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',  
 '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',  
 '__subclasshook__', '__weakref__', 'get_fabricante', 'set_fabricante']
```

```
print(carro.__fabricante__)
```

AttributeError: 'Carro' object has no attribute '__fabricante__'

```
print(carro.__fabricante)
```

AttributeError: 'Carro' object has no attribute '__fabricante'

```
print(carro._Carro__fabricante)
```

Toyota

Encapsulamento, atributos públicos e privados.

Campos “privados” não podem ser acessados por uma subclasse.

```
class Mae(object):
    def __init__(self):
        self.__atributo_privado = 10

class Filha(Mae):
    def pegar_atributo_privado(self):
        return self.__atributo_privado

filha = Filha()
print(filha.__dict__)
x = filha.pegar_atributo_privado()
print(x)
```

`__dic__`:
Retorna um
dicionário
com os
atributos do
objeto.

```
{'_Mae__atributo_privado': 10}
return self.__atributo_privado
AttributeError: 'Filha' object has no attribute '_Filha__atributo_privado'
```

```
class Mae(object):
    def __init__(self):
        self.__atributo_privado = 10

class Filha(Mae):
    def pegar_atributo_privado(self):
        # return self.__atributo_privado
        return self._Mae__atributo_privado

filha = Filha()
x = filha.pegar_atributo_privado()
print(x)
```

Encapsulamento, atributos públicos e privados.

“Por que então a sintaxe para atributos privados não assegura as restrições de visibilidade como deveria? A resposta mais simples é um dos lemas mais citados do Python: ‘Somos todos adultos aqui e consentimos em dar liberdade uns aos outros’. Os programadores de Python acreditam que os benefícios da liberdade são maiores que as desvantagens de ter a cabeça fechada.”

SLATKIN, B. *Python Eficaz*. São Paulo: Novatec, 2016. 121 p.

CONTINUA...