



Prof. Dr. Fernando Almeida
proffernando.almeida@fiap.com.br

DDD (Domain Driven Design)

Métodos Acessores e Modificadores em Java

Métodos Construtores

O QUE VAMOS APRENDER HOJE?

1

Getters

2

Setters

3

Acessores e modificadores em classes
publicas

4

Acessores e modificadores em classes
privadas

5

Métodos Construtores

Getters e Setters em Java

Definição

A photograph of Joshua Bloch, a man with grey hair and glasses, wearing a black t-shirt with the word "Google" in its signature multi-colored font. He is looking down at a white smartphone he is holding in his hands. The background is blurred, showing some yellow flowers and greenery.

“O fator mais importante que distingue um módulo bem projetado de um mal projetado é o grau com que o módulo oculta de outros módulos os seus dados internos e outros detalhes de implementação.

- Joshua Bloch

“ Uma das características essenciais de projetar um certo módulo é definir a sua capacidade de ocultar todos os seus detalhes de implementação... ”

Métodos *get* e *set*

- São técnicas padronizadas para gerenciar o acesso aos atributos
- Acesso aos atributos **privados**
- Definição da forma de alteração e acesso aos atributos
- Tornam o controle e modificações dos atributos mais práticas e limpas, sem precisar alterar a assinatura do método usado para acesso ao atributo

Métodos Acessores

Métodos *get*

Método *get()*

String s = <obj>.getNomeProduto();

- Utilizados para acessar, “pegar” o conteúdo dos atributos da classe
- Esse método sempre retornará um valor (String, int, float, double, char, etc)
- Dentro desse método haverá somente o retorno do atributo

```
1 public String getNomeProduto() {  
2     return nomeproduto;  
3 }  
4  
5 public int getQuantidade() {  
6     return quantidade;  
7 }  
8  
9 public String getValorUnitario() {  
10    return valorunitario;  
11 }
```

Métodos Acessores

Métodos *set*

Método set()

- Utilizados para alterar, modificar o conteúdo dos atributos da classe de forma **protegida**
- Esse método **não** terá retorno (tipo **void**)
- Haverá apenas a modificação do atributo
- Deve receber algum argumento para a devida alteração

```
1 | public void setNomeProduto(String nomeproduto) {
2 |     this.nomeproduto = nomeproduto;
3 | }
4 |
5 | public void setQuantidade(int quantidade) {
6 |     this.quantidade = quantidade;
7 | }
8 |
9 | public void setValorUnitario(String valorunitario) {
10|    this.valorunitario = valorunitario;
11| }
```

Observação: A utilização da cláusula THIS faz referência ao atributo da classe dentro da qual se está trabalhando

Acessores e Modificadores

Classes Púlicas

“ Quando temos uma classe pública com seus atributos sendo diretamente acessados, dizemos que ela **não** oferece os benefícios do **encapsulamento**

Encapsulamento

- Torna o software mais flexível, fácil de modificar e criar novas implementações
- Controle de acesso aos atributos e métodos de uma classe
- Forma eficiente de proteger os dados manipulados dentro da classe
- Determina onde a classe poderá ser manipulada

Exemplo de uma classe mal projetada

- Exposição direta às instâncias

```
1 | public class Ponto {  
2 |     public double x;  
3 |     public double y;  
4 | }
```

- Não há uma forma de tomar medidas quando o campo é acessado
- Exemplo: uma nova regra que um determinado campo deve ser somado 5 ao seu valor.
- Falta de controle (manipulação por algum acesso externo)

Classe encapsulada com métodos acessores e modificadores

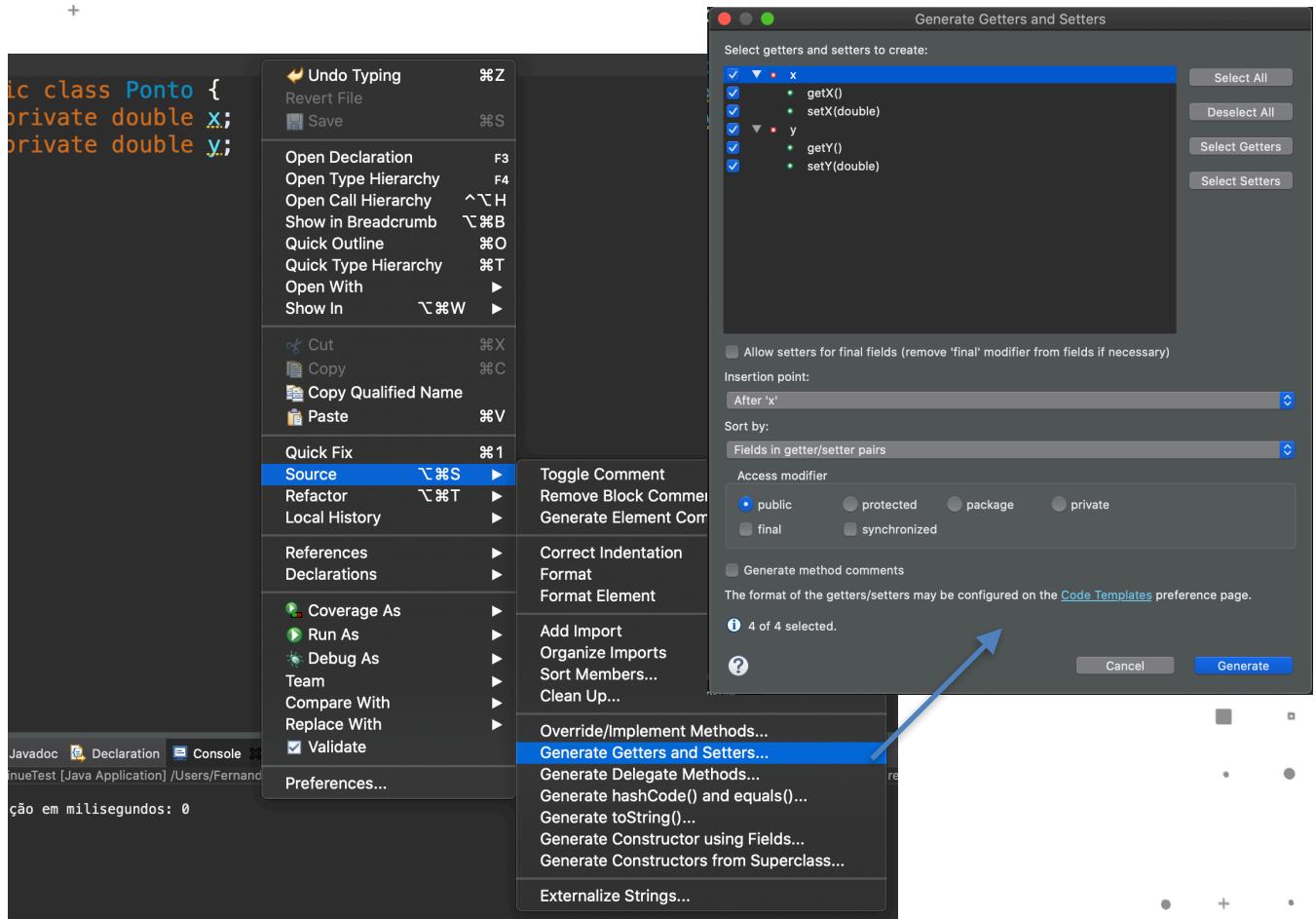
- **Regra sobre classes públicas**
 - Deve-se sempre fornecer **campos privados** e **métodos acessores (getters) públicos**
- **Regra sobre classes mutáveis**
 - Deve-se fornecer os métodos modificadores (**setters**)

Classe encapsulada com métodos acessores e modificadores

```
1 public class Ponto {  
2     private double x;  
3     private double y;  
4  
5     public Ponto(double x, double y) {  
6         this.x = x;  
7         this.y = y;  
8     }  
9  
10    public double getX() { return x; }  
11  
12    public double getY() { return y; }  
13  
14    public void setX(double x) { this.x = x; }  
15  
16    public void setY(double y) { this.y = y; }  
17  
18 }
```

Método Construtor

- Após declarar os atributos, clique com o botão direito do mouse dentro da classe
- Selecione a opção **Source**
- Selecione **Generate Getters and Setters**



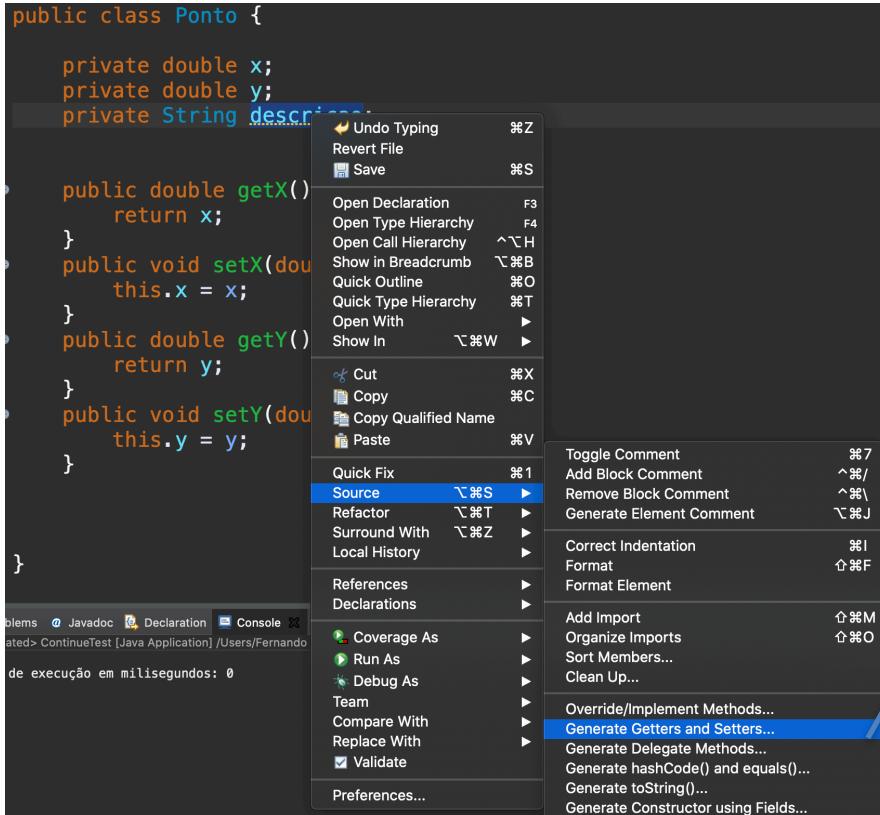
Classe com getters e setters (gerados automaticamente)

```
1 | public class Ponto {  
2 |  
3 |     private double x;  
4 |     private double y;  
5 |  
6 |     public double getX() {  
7 |         return x;  
8 |     }  
9 |     public void setX(double x) {  
10 |         this.x = x;  
11 |     }  
12 |     public double getY() {  
13 |         return y;  
14 |     }  
15 |     public void setY(double y) {  
16 |         this.y = y;  
17 |     }  
18 | }  
19 }
```

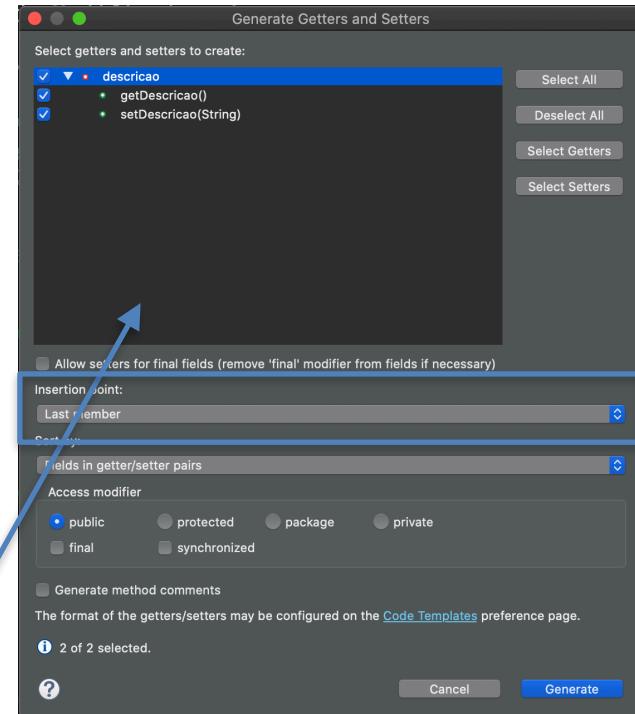
Classe com getters e setters (com comentários)

```
1 | public class Ponto {  
2 |  
3 |     private double x;  
4 |     private double y;  
5 |  
6 |     /**  
7 |     * @return the x  
8 |     */  
9 |     public double getX() {  
10 |         return x;  
11 |     }  
12 |     /**  
13 |     * @param x the x to set  
14 |     */  
15 |     public void setX(double x) {  
16 |         this.x = x;  
17 |     }  
18 |     /**  
19 |     * @return the y  
20 |     */  
21 |     public double getY() {  
22 |         return y;  
23 |     }  
24 |     /**  
25 |     * @param y the y to set  
26 |     */  
27 |     public void setY(double y) {  
28 |         this.y = y;  
29 |     }  
30 | }
```

Classe com getters e setters (para campos exclusivos)



```
public class Ponto {  
  
    private double x;  
    private double y;  
    private String descricao;  
  
    public double getX() {  
        return x;  
    }  
    public void setX(double x) {  
        this.x = x;  
    }  
    public double getY() {  
        return y;  
    }  
    public void setY(double y) {  
        this.y = y;  
    }  
}
```



Acessores e Modificadores

Classes Privadas de Pacote

Acessores e modificadores para classes Privadas de Pacote

- As vezes é desejável expor os campos (mutável ou imutável) de classes privadas de pacotes ou classes aninhadas privadas
- Se tivermos uma classe privada de pacote (default, sem modificador de acesso) ou então uma classe aninhada privada, não há problema em expor seus dados
- Gera menos confusão visual do que a abordagem de métodos acessores tanto na definição da classe quanto no código "clientes" que a usa (vinculado à representação interna da classe)
- O cliente está vinculado ao pacote em que a classe está contida
- Se a classe for aninhada privada, o escopo de uma alteração é ainda mais restrita, envolvendo apenas a classe delimitadora

Métodos em Java

Métodos Construtores

“Em Java, todas as classes, incluindo as classes abstratas, necessitam pelo menos de um **construtor**. Através dele o desenvolvedor terá acesso à classe, aos atributos e métodos. Sempre serão chamados em tempo de execução e disponibilizarão a criação de um objeto da classe”

“ (...), o **construtor** é definido como um método cujo nome deve ter o mesmo nome da classe e sem indicação do tipo retorno (inclusive **void**). É unicamente invocado no momento da criação do objeto através do operador **new**. E o retorno é uma referência para o objeto recém-criado (**cria o objeto em memória**)

Declaração de Construtores

```
1 public class Carro{  
2     /* CONSTRUTOR DA CLASSE Carro */  
3     public Carro(){  
4         //Faça o que desejar na construção do objeto  
5     }  
6 }  
7 }  
8 }
```

O construtor pode ter níveis como: **public, private ou protected**

Mas porque definiríamos um construtor como **privado**?

Chamando um Construtor da Classe Carro

```
1 public class Carro{  
2  
3     /* CONSTRUTOR DA CLASSE Carro */  
4     public Carro(){  
5         //Faça o que desejar na construção do objeto  
6     }  
7  
8 }
```

Criado por padrão

```
9  
10 public class Aplicacao {  
11  
12  
13     public static void main(String[] args) {  
14         //Chamamos o construtor sem nenhum parâmetro  
15         Carro fiat = new Carro();  
16     }  
17  
18 }
```

Definindo vários construtores

```
1 public class Carro{  
2  
3     private String cor;  
4     private double preco;  
5     private String modelo;  
6  
7     /* CONSTRUTOR PADRÃO */  
8     public Carro(){  
9         //  
10    }  
  
11  
12    /* CONSTRUTOR COM 2 PARÂMETROS */  
13    public Carro(String modelo, double preco){  
14        //Se for escolhido o construtor sem a COR do veículo  
15        // definimos a cor padrão como sendo PRETA  
16        this.cor = "PRETA";  
17        this.modelo = modelo;  
18        this.preco = preco;  
19    }  
  
20  
21    /* CONSTRUTOR COM 3 PARÂMETROS */  
22    public Carro(String cor, String modelo, double preco){  
23        this.cor = cor;  
24        this.modelo = modelo;  
25        this.preco = preco;  
26    }  
27  
28 }
```

Temos agora 3 construtores padrão

Criando vários carros com diferentes Construtores

```
1 public class Carro{  
2  
3     private String cor;  
4     private double preco;  
5     private String modelo;  
6  
7     /* CONSTRUTOR PADRÃO */  
8     public Carro(){  
9  
10    }  
11  
12    /* CONSTRUTOR COM 2 PARÂMETROS */  
13    public Carro(String modelo, double preco){  
14        //Se for escolhido o construtor sem a COR do veículo  
15        // definimos a cor padrão como sendo PRETA  
16        this.cor = "PRETA";  
17        this.modelo = modelo;  
18        this.preco = preco;  
19    }  
20  
21    /* CONSTRUTOR COM 3 PARÂMETROS */  
22    public Carro(String cor, String modelo, double preco){  
23        this.cor = cor;  
24        this.modelo = modelo;  
25        this.preco = preco;  
26    }  
27  
28 }
```

```
30 public class Aplicacao {  
31  
32  
33     public static void main(String[] args) {  
34         //Construtor sem parâmetros  
35         Carro prototipoDeCarro = new Carro();  
36  
37         //Construtor com 2 parâmetros  
38         Carro civicPreto = new Carro("New Civic", 40.000);  
39  
40         //Construtor com 3 parâmetros  
41         Carro golfAmarelo = new Carro("Azul", "Golf", 38.000);  
42     }  
43  
44 }
```

Chamada de 3 construtores

Classe com construtor implícito

- Construtor implícito (criado pelo compilador) - não visível no código fonte

```
2
3  public class Animal {
4      private double peso;
5      private String grupo;
6      // gerar getters e setters
7      public static void main(String[] args) {
8          Animal animal = new Animal();
9          animal.setPeso(5.5);
10         animal.setGrupo("Mamiferos");
11         System.out.println("Peso: " + animal.getPeso() +
12             " Grupo: " + animal.getGrupo());
13     }
14 }
```

Será preciso gerar os
getters e setters

Classe com construtor explícito

- Construtor explícito (criado pelo desenvolvedor) - visível no código fonte

```
3 public class Animal {  
4     private double peso;  
5     private String grupo;  
6     //gerar getters and setters  
7  
8     public Animal() {  
9         //padrão  
10    }  
11  
12    public Animal(double peso, String grupo) {  
13  
14        this.peso = peso;  
15        this.grupo = grupo;  
16    }  
17  
18    public static void main(String[] args) {  
19        Animal a = new Animal();  
20        a.setPeso(5.5);  
21        a.setGrupo("Mamíferos");  
22        System.out.println("Peso: " + a.getPeso() + " Grupo: " + a.getGrupo());  
23        Animal b = new Animal(6.0, "Aves");  
24        System.out.println("Peso: " + b.getPeso() + " Grupo: " + b.getGrupo());  
25    }  
26}
```



Métodos em Java

Métodos Destruidores



Métodos destrutores em Java

- Não existe o conceito de destrutores em Java
- Não tem como “destruir” um objeto (assim como se faz em C/C++)
- Não há garantia de que o **Garbage Collector** irá destruir um objeto (trabalha por conveniência) - sem controle pelo programador!
- A forma mais adequada de “tentar” destruir um objeto em Java é atribuir valores **nulos** a ele

OBRIGADO

FIAP

Copyright © 2021 | Prof. Dr. Fernando Luiz de Almeida

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente
proibido sem consentimento formal, por escrito, do professor/autor.



Até a próxima aula