



# Semana 3

Construtores & Encapsulamento

Prof.<sup>a</sup> Marina de Lara



01

# Construtores

# Construtores

```
public class Main {  
    Pessoa pessoa = new Pessoa();  
    pessoa.nome = "Marina"  
    pessoa.idade = 31  
}
```

Até agora nós aprendemos a criar classes dessa forma. Usamos o comando **new** para **instanciar** um objeto e atribuímos os valores dos atributos acessando-os diretamente.

# Construtores

```
public class Main {  
    Pessoa pessoa = new Pessoa();  
    pessoa.nome = "Marina"  
    pessoa.idade = 31  
}
```

Até agora nós aprendemos a criar classes dessa forma. Usamos o comando **new** para **instanciar** um objeto e atribuímos os valores dos atributos acessando-os diretamente.

## Mas como exatamente o objeto está sendo criado?

# Construtores

```
Pessoa pessoa = new Pessoa();
```

Em linguagens orientadas a objetos, como o Java, nós criamos um objeto a partir de um **construtor**. O construtor da classe é responsável por criar aquela “caixinha” na memória que irá armazenar todos os dados referentes ao objeto que foi criado.

# Construtores

```
Pessoa pessoa = new Pessoa();
```

Por padrão, quando não definimos um construtor para classe, o Java cria um construtor automaticamente. Nós chamamos esse construtor de **construtor vazio** ou **construtor em branco**. Esse construtor não é visível na implementação, pois é criado de forma implícita pelo Java.



**Mas será que conseguimos  
modificar um construtor?**





# SIM!

Podemos criar diversos tipos de construtores para atender melhor as necessidades das nossas classes.

Chamamos esse tipo de construtor de **construtor explícito**.





# Construtores

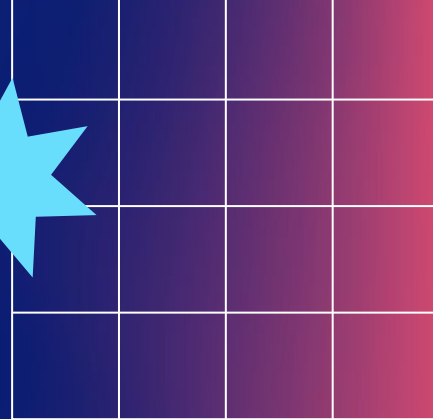
```
public class Pessoa {  
    public Pessoa() {  
        //Código aqui  
    }  
}
```

Para criar um construtor dentro de uma classe, utilizamos um modificador de acesso (iremos ver isso daqui a pouco), nesse caso a palavra `public`, seguido do nome da própria classe. **O nome do construtor precisa ser IGUAL ao da classe.**

# Construtores

```
public class Pessoa {  
    public Pessoa() {  
        //Código aqui  
    }  
}
```

Reparem que no construtor nós não colocamos um tipo de retorno, pois um construtor **nunca irá retornar nada**. Lembre que ele é responsável apenas por instanciar (criar) o nosso objeto na memória.



# Construtores

A implementação da classe pessoa, levando em consideração apenas os atributos e o construtor (sem métodos), poderia ser feita da seguinte forma:

```
public class Pessoa {  
    public String nome;  
    public int idade;  
  
    public Pessoa() {  
        this.nome = "Usuário";  
        this.idade = 0;  
    }  
}
```

# Construtores

Com essa implementação de construtor, todo objeto do tipo Pessoa que criamos no nosso programa, vai ter por padrão o nome “Usuário” e a idade 0. Esses valores podem ser modificados posteriormente.

```
public class Pessoa {  
    public String nome;  
    public int idade;  
  
    public Pessoa() {  
        this.nome = “Usuário”;  
        this.idade = 0;  
    }  
}
```

# Construtores

Outro tipo de construtor que conseguimos criar é o construtor com parâmetros de entrada. Podemos obrigar que alguns dados sejam fornecidos no momento em que o objeto for instanciado.

```
public class Pessoa {  
    public String nome;  
    public int idade;  
  
    public Pessoa(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
}
```

# Construtores

No momento que implementamos um construtor explícito com parâmetros de entrada, nós não conseguimos mais instanciar os objetos sem fornecer esses parâmetros.

**Esse não funciona mais**

```
Pessoa pessoa = new Pessoa();
```

**Esse aqui funciona**

```
Pessoa pessoa = new Pessoa("Marina", 31);
```

# Construtores

```
public class Pessoa {  
    public String nome;  
    public int idade;  
  
    public Pessoa() {  
        this.nome = "Usuário";  
        this.idade = 0;  
    }  
  
    public Pessoa(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
}
```

Nós podemos implementar mais de um construtor na mesma classe, dessa forma podemos escolher qual dos construtores queremos usar. Isso é útil quando temos diferentes situações para um mesmo objeto.

# Construtores

No caso de termos implementado ambos os construtores na classe, podemos utilizar os dois formatos para construir um objeto.

## Esse aqui funciona

```
Pessoa pessoa = new Pessoa();
```

## Esse aqui também

```
Pessoa pessoa = new Pessoa("Marina", 31);
```



# Construtores

```
public class Pessoa {  
    public String nome;  
    public int idade;  
  
    public Pessoa() {  
        this("Usuário", 0);  
    }  
  
    public Pessoa(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
}
```

Uma outra forma de atribuir valores aos atributos no construtor sem parâmetros, seria chamar o construtor com parâmetros declarado logo abaixo. Isso **não irá alterar** a forma como instanciamos o objeto.

# this

A palavra chave **this** do Java é utilizada quando estamos nos referindo a atributos, construtores e métodos da própria classe.


Seu uso é opcional quando não há ambiguidade no nome das palavras:

```
public class Pessoa {  
    public String nome;  
  
    public Pessoa(String texto) {  
        nome = texto;  
    }  
}
```

# this

Mas é obrigatório quando chamamos o construtor da classe dentro dela mesma e quando o nome dos atributos da classe é o mesmo nome do valor que recebemos por parâmetro:

```
public class Pessoa {  
    public String nome;  
  
    public Pessoa() {  
        this("Usuário");  
    }  
  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
}
```



**Na dúvida, use o `this` para  
atributos e parâmetros  
dentro da classe.**

**:)**



# 02 Encapsulation



# Encapsulamento

Até agora nós utilizamos a palavra **public** sem saber exatamente o que ela significa e se pode ser alterada. Para entender melhor a sua utilização, vamos entender primeiro o que é encapsulamento.

**Encapsulamento** é uma técnica utilizada para **omitir atributos** e **limitar o seu acesso**, tornando-os ocultos para outros objetos.

Com isso, a responsabilidade de como as coisas são definidas e implementadas dentro de uma classe, passa a ser **exclusiva da classe**, e não mais de objetos externos à ela.

# Modificadores de Acesso

Para entender melhor as várias camadas do encapsulamento, vamos começar falando de **modificadores de acesso**.

Existem três tipos de **modificadores de acesso**:



**public**



**private**



**protected**



Os atributos e métodos definidos como **public** podem ser acessados por qualquer classe

**public**

Os atributos e métodos definidos como **private** só podem ser acessados dentro da classe em que foram declarados



**private**



Os atributos e métodos definidos como **protected** podem ser acessados dentro da própria classe e das subclasses

**protected**

Vamos falar disso mais pra frente



# Encapsulamento

```
public class Pessoa {  
    private String nome;  
    private int idade;  
}
```

O princípio mais básico do encapsulamento é que todos os atributos de uma classe devem **sempre** utilizar o modificador de acesso **private** ou **protected**.

**Por enquanto vamos utilizar apenas o private**

# Encapsulamento

```
public class Pessoa {  
    private String nome;  
    private int idade;  
}
```

Beleza, temos os nossos dois atributos encapsulados. Mas e agora como fazemos para permitir que outras classes acessem esses atributos? E se quisermos alterar o nome ou imprimir esse nome fora da classe?



# Getters e Setters

Aqui vamos para a segunda camada do encapsulamento. Para permitir que os atributos sejam acessados fora da classe, iremos utilizar métodos chamados de **getters** e **setters**.

## Getters:


Permite que o valor de um atributo seja obtido fora da classe.

Podemos pensar nos getters como métodos de leitura de atributos.

## Setters:

Permite que o valor de um atributo seja modificado fora da classe.

Podemos pensar nos setters como métodos de modificação de atributos.



# Getters

## Getters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
}
```

Como vimos anteriormente, o **modificador de acesso public** permite que atributos e métodos sejam acessados fora da classe, portanto se queremos **permitir** que outras classes obtenham o valor dos atributos da classe **Pessoa**, é preciso utilizar o **public** nesses métodos.

# Getters

## Getters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
}
```

Além disso, os métodos **getters** sempre utilizam o tipo de retorno **igual ao atributo** que está retornando. No exemplo, o **getter** do atributo **nome** possui retorno do tipo **String**, enquanto o **getter** de **idade** possui seu retorno do tipo **int**.

# Getters

## Getters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public String getNome() {  
        return nome;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
}
```

É comum que o nome de um método **getter** comece com a palavra **get** seguido do **nome do atributo** que está retornando.

É possível implementar mais funcionalidades dentro de um método **getter** caso seja necessário, mas por padrão deve sempre **retornar o atributo** em questão.

# Getters

## Getters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    private boolean dormindo;  
  
    public boolean isDormindo() {  
        return dormindo;  
    }  
}
```

Uma exceção à nomenclatura do nome do método **getter** é quando estamos retornando um parâmetro **boolean**. Nesse caso o nome do **getter** começa com **is** seguido do nome do atributo, como no exemplo ao lado.

# Setters

## Setters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public void setIdade(int idade) {  
        this.idade = idade;  
    }  
}
```

Da mesma forma que o **getter**, o **setter** também precisa utilizar o **modificador de acesso public** para **permitir** que outras classes possam **alterar o valor** de um determinado atributo da classe **Pessoa**.



# Setters

## Setters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public void setIdade(int idade) {  
        this.idade = idade;  
    }  
}
```

Percebam que nos **setters** o retorno sempre será do tipo **void**, pois o objetivo de um método **setter** não é retornar nenhum valor para outras classes, mas sim **alterar o valor** de um atributo para um **novo valor** recebido por parâmetro.

# Setters

## Setters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public void setIdade(int idade) {  
        this.idade = idade;  
    }  
}
```

É comum que o nome do método utilize a palavra **set** seguido do **nome do atributo** que está sendo modificado. Além disso, é preciso **sempre** receber como parâmetro o **novo valor** que será informado fora da classe, de outra forma não conseguimos modificar o valor do atributo.

# Setters



## Setters:

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public void setIdade(int idade) {  
        if(idade ≥ 0) {  
            this.idade = idade;  
        } else {  
            System.out.println("Idade não  
pode ser menor que 0");  
        }  
    }  
}
```

A implementação do método pode ser **simples**, apenas atribuindo o novo valor ao atributo em questão, ou podemos implementar algumas **regras de atribuição** de um valor. No exemplo do método **setIdade**, uma condição foi implementada de forma que o atributo idade só pode ser modificado caso seja maior ou igual a 0, do contrário exibe uma mensagem de erro no console.

# Encapsulament



Não somos obrigados a implementar **getters** e **setters** para todos os nossos atributos, isso fica a critério de como o seu software está sendo modelado.

Quem está fazendo o software que precisa definir quais atributos podem ser acessados ou não por outras classes.

Um exemplo disso seria deixar a atribuição dos valores apenas para o **construtor** do objeto e implementar apenas **getters** para permitir a leitura dos valores fora da classe.

```
public class Pessoa {  
    private String nome;  
    private int idade;  
  
    public Pessoa(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
}
```

# Utilização

Um exemplo de como seria o código na classe Main. Primeiro criamos um objeto do tipo

**Pessoa** utilizando o construtor explícito da classe, que recebe como parâmetro o **nome** e a **idade** da pessoa. Em seguida imprimimos o **nome** e a **idade** utilizando os métodos **getters**.

Por fim, alteramos o valor do **nome** e **idade** utilizando os **setters** e imprimimos novamente utilizando os **getters**.

```
public class Main {  
  
    //Instanciando objeto Pessoa  
    Pessoa pessoa = new Pessoa("Marina", 31);  
  
    //Imprimindo informações com getters  
    System.out.println(pessoa.getNome());  
    System.out.println(pessoa.getIdade());  
  
    //Alterando informações com setters  
    pessoa.setNome("Marina de Lara");  
    pessoa.setIdade(32);  
  
    //Imprimindo informações com getters  
    System.out.println(pessoa.getNome());  
    System.out.println(pessoa.getIdade());  
  
}
```

**Podemos utilizar private em método?**



**Podemos utilizar private em  
método?**

**SIM!**

**Pra que?**

# Métodos Encapsulados

Métodos privados encapsulam funcionalidades que são utilizadas apenas dentro da classe, mas não podem ser acessados por outras classes ou partes do código. Esses métodos são conhecidos como **métodos auxiliares**.

```
public class Pessoa {  
    private String nome;  
    private int idade;  
    private int codigo;  
  
    public Pessoa(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
        this.codigo = gerarCodigo();  
    }  
  
    private int gerarCodigo() {  
        Random random = new Random();  
        return random.nextInt(9000) + 1000;  
    }  
}
```



# Para Finalizar...



Todos os lugares que vemos a palavra **public** pode ser substituída por **private** ou **protected**, incluindo classes e construtores. Mas por enquanto não vamos complicar muito.

Lembretes de **Boas Práticas**:

1. **TODOS** os atributos devem ser encapsulados (private);
2. Faça uma análise de quando é necessário implementar **getters** e **setters**, na dúvida implemente;
3. O nome dos métodos **getter** é composto da seguinte forma: **get + nome do atributo**;
4. O nome dos métodos **setter** é composto da seguinte forma: **set + nome do atributo**;
5. O nome dos métodos **getter** do tipo **booleano** é composto da seguinte forma: **is + nome do atributo**;
6. Na dúvida, utilize sempre o **this** quando estiver utilizando atributos e métodos internos.
7. Sempre importante lembrar: **Nome de classe sempre com a PRIMEIRA LETRA MAIÚSCULA**. O restante do código segue sempre o padrão **camelCase**.




# 03

# Atividade



**Utilize o modelo criado na aula passada e implemente todas as classes em Java da seguinte forma:**

1. Todas as classes devem possuir 2 construtores explícitos, um com e um sem parâmetros;
2. Todos os atributos devem seguir os princípios de encapsulamento, fica ao critério da sua equipe quais deles devem possuir getters e setters.



**Para as equipes que já possuem o código pronto:** revisem se está tudo de acordo com o que foi pedido acima. Caso esteja tudo certo, escolha outro sistema para fazer a atividade.

**Entrega: Quinta-feira 08/03 - até 08:00**

