07

Para saber mais: reaproveitamento entre construtores

Nesse capítulo o nosso aprendizado foi focado nos construtores. Eles são elaborados visando que os objetos tenham seus atributos inicializados na própria construção. Essa estratégia evita estados inconsistentes no nosso objeto. Veja essa classe:

```
public class Carro{
    private int ano;
    private String modelo;
    private double preco;

    //getters e setters omitidos

}
```

Como já se sabe, quando o construtor não está declarado na classe usa-se o padrão, que não recebe parâmetro algum. Logo, uma utilização da classe poderia ser como a seguir:

```
Carro carro = new Carro();
carro.setAno(2013);
carro.setPreco(35000.0);
```

Ficou faltando uma informação preciosa! Qual o modelo dele? Para evitar esse tipo de problema devemos exigir os dados que fazem sentido o Carro ter logo na criação. Algo como:

```
public class Carro{
    private int ano;
    private String modelo;
    private double preco;

    public Carro(int ano, String modelo, double preco){
        this.ano = ano;
        this.modelo = modelo;
        this.preco = preco;
    }

    //getters e setters omitidos

}
```

Agora a utilização exige a presença dos 3 parâmetros definidos.

```
Carro carro = new Carro(2013, "Gol", 35000.0);
```

Tudo funciona bem! Até que um dia é pedido que o nosso sistema aceite a criação com a passagem somente do modelo e valor. Nessa situação deve-se encarar o ano como sendo 2017. Uma solução seria:

```
public class Carro{
    private int ano;
    private String modelo;
    private double preco;

    public Carro(int ano, String modelo, double preco){
        this.ano = ano;
        this.modelo = modelo;
        this.preco = preco;
    }

    //Novo construtor AQUI!
    public Carro(String modelo, double preco){
        this.ano = 2017;
        this.modelo = modelo;
        this.preco = preco;
    }

    //getters e setters omitidos

}
```

E dessa forma pode-se construir carros com qualquer um dos dois construtores:

```
Carro carro = new Carro(2013, "Gol", 35000.0);
Carro outroCarro = new Carro("Civic", 95000.0);
```

Só que na empresa onde esse sistema está sendo codificado existe uma equipe de testes que verificou que o nosso sistema permite a criação de um Carro com datas anteriores ao primeiro automóvel que chegou ao Brasil, um Peugeot trazido por Santos Dumont em 1891. (Alura também é história!) Além de também permitir que o modelo não seja passado(null) e o preço inválido.

O desenvolvedor logo tratou de implementar essa regra em um dos construtores:

```
public class Carro{
    private int ano;
    private String modelo;
    private double preco;

    public Carro(int ano, String modelo, double preco){
        if(ano >= 1891){
            this.ano = ano;
        }else{
            System.out.println("O ano informado está inválido. Por isso usaremos 2017!");
            this.ano = 2017;
```

```
        }

        if( modelo != null){
            this.modelo = modelo;
        }else{
            System.out.println("O modelo não foi informado. Por isso usaremos Gol!");
            this.modelo = "Gol";
        }

        if(preco > 0){
            this.preco = preco;
        }else{
            System.out.println("O preço não é válido. Por isso usaremos 40000.0!");
            this.preco = 40000.0;
        }
    }
    //....

}
```

Perceba que como temos dois construtores a regra também deveria valer para o outro:

```
public class Carro{
    private int ano;
    private String modelo;
    private double preco;

    public Carro(int ano, String modelo, double preco){
        if(ano >= 1891){
            this.ano = ano;
        }else{
            System.out.println("O ano informado está inválido. Por isso usaremos 2017!");
            this.ano = 2017;
        }

        if( modelo != null){
            this.modelo = modelo;
        }else{
            System.out.println("O modelo não foi informado. Por isso usaremos Gol!");
            this.modelo = "Gol";
        }

        if(preco > 0){
            this.preco = preco;
        }else{
            System.out.println("O preço não é válido. Por isso usaremos 40000.0!");
            this.preco = 40000.0;
        }

    }

    //Novo construtor AQUI!
```

```
    public Carro(String modelo, double preco){
        this.ano = 2017;
        if( modelo != null){
            this.modelo = modelo;
        }else{
            System.out.println("O modelo não foi informado. Por isso usaremos Gol!");
            this.modelo = "Gol";
        }

        if(preco > 0){
            this.preco = preco;
        }else{
            System.out.println("O preço não é válido. Por isso usaremos 40000.0!");
            this.preco = 40000.0;

    }

    //getters e setters omitidos

}
```

Funcionou mas o código está duplicado e nossa classe começa a cheirar mal! Códigos duplicados exigem manutenção em dobro no futuro e em grande parte das vezes um futuro nem tão distante. Seria ótimo se fosse possível reaproveitar a lógica de validação do primeiro construtor declarado não é mesmo? Reaproveitaríamos todo ele e qualquer mudança também traria o impacto direto. No Java podemos chamar a implementação de um construtor através de outro usando simplesmente `this( )` com os parâmetros exigidos pelo construtor.

Observe como ficaria o segundo construtor da nossa classe:

```
    public Carro(String modelo, double preco){
    //chamando o construtor que recebe int, String e double. Nosso primeiro!
        this(2017, modelo, preco);
    }
```

Muito mais simples de manter não é mesmo? Nossa classe, Carro, ficaria portanto assim:

```
public class Carro{
    private int ano;
    private String modelo;
    private double preco;

    public Carro(int ano, String modelo, double preco){
        if(ano >= 1891){
            this.ano = ano;
        }else{
            System.out.println("O ano informado está inválido. Por isso usaremos 2017!");
            this.ano = 2017;
        }

        if( modelo != null){
```

```
            this.modelo = modelo;
        }else{
            System.out.println("O modelo não foi informado. Por isso usaremos Gol!");
            this.modelo = "Gol";
        }

        if(preco > 0){
            this.preco = preco;
        }else{
            System.out.println("O preço não é válido. Por isso usaremos 40000.0!");
            this.preco = 40000.0;
        }
    }

    //Novo construtor AQUI!
    public Carro(String modelo, double preco){
        this(2017, modelo, preco);
    }

    //getters e setters omitidos

}
```

Conclusão

No Java é possível fazer a chamada de um construtor dentro de outro e faz-se isso para evitar duplicações de códigos e regras. Afinal uma regra aplicada em um construtor normalmente será a mesma para o outro caso. Para isso usa-se o `this()` passando os parâmetros correspondentes ao construtor que se queira chamar.