Roteiro 5

Sobrecarga de Métodos

Neste roteiro utilizaremos um novo cenário, dando sequência aos conhecimentos sobre OO adquiridos no roteiro 4, e faremos a introdução de um novo conceito : Sobrecarga.

A **sobrecarga** (*overload*) consiste em permitir, dentro da mesma classe, mais de um método com o mesmo nome. Normalmente acontece sobre os métodos construtores, pois é comum para uma classe ter várias maneiras de instanciá-la.

Cenário :

Deseja-se um controle simples de informações cadastrais de uma loja de roupas que possui um nome de fantasia, uma razão social, um número de CNPJ, o valor de faturamento e a área em m^2 que ela ocupa, e o nome do proprietário.

Parte 1 (roteiro5.parte1)

1 – Crie o pacote **roteiro5.parte1** com a classes **Principal** e **Loja** . Conforme o cenário descrito, teremos que criar os seguintes atributos para a classe Loja :

- String nomeFantasia
- String razaoSocial
- String cnpj
- double valorFat
- double área
- String nomeProprietario

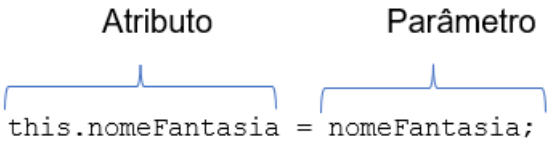
Crie os atributos indicados na classe Loja com o acesso private.

2 – Implemente os métodos Gets e Sets da classe Loja.

3 – Agora iremos implementar o método construtor de forma que ele receba como parâmetro o nome de Fantasia, a Razão Social e o CNPJ.

Observe que no código abaixo utilizamos a palavra reservada **this** para acessar os atributos. Sempre que estamos implementando qualquer código dentro de uma classe específica (neste caso Loja) e queremos acessar um atributo ou método, podemos usar this conforme o exemplo abaixo (com exceção os atributos ou métodos estáticos - static).

Neste caso, o **this** também foi útil para diferenciar as variáveis recebidas como parâmetro no construtor, dos atributos da classe




```
public class Loja {
    private String nomeFantasia;
    private String razaoSocial;
    private String cnpj;
    private double valorFat;
    private double area;

    public Loja(String nomeFantasia, String razaoSocial, String cnpj){
        this.nomeFantasia = nomeFantasia;
        this.razaoSocial = razaoSocial;
        this.cnpj = cnpj;
    }

    {
      Implementar Gets e Sets
    }

}
```

4 – Precisamos testar o funcionamento da classe Loja. crie na classe Principal o objeto loja01 e faça os devidos testes.

Observe que alguns atributos não foram passados no construtor. O que acontece quando exibimos os dados da loja01 ? Atenção para valores null.

```
public class Principal {

    public static void main(String[] args) {

      Loja loja01 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");

    }
}
```

5 - No momento do cadastro da loja nem sempre é possível ter as 3 informações do construtor (Nome de Fantasia, Razão Social e CNPJ). Na maioria das vezes o usuário possui apenas o Nome de Fantasia e o CNPJ. Desta forma, definimos a seguinte regra de negócio : Sempre que não tivermos a Razão Social, iremos preencher este atributo com o Nome de Fantasia.

Trata-se de um problema relativamente simples de resolver, basta testar o parâmetro razaoSocial no construtor da classe Loja

```
public class Loja {
    private String nomeFantasia;
    private String razaoSocial;
    private String cnpj;
    private double valorFat;
    private double area;

    public Loja(String nomeFantasia, String razaoSocial, String cnpj){
        this.nomeFantasia = nomeFantasia;

        if (razaoSocial.equals("")){
            this.razaoSocial = nomeFantasia;
        }
        else {
            this.razaoSocial = razaoSocial;
        }

        this.cnpj = cnpj;
    }

    {
      Implementar Gets e Sets
    }
}
```

6 – Para testar a solução criamos 2 objetos do tipo Loja na classe Principal (loja01 e loja02). A loja02 não tem a Razão Social. Ao imprimir os dados das duas lojas, tivemos o resultado esperado ? A solução implementada no item 5 resolveu o problema ?

```
public class Principal {

    public static void main(String[] args) {

      Loja loja01 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");

      Loja loja02 = new Loja("Mercadão do Povo", "", "10101010");

    }
}
```

Parte 2 (roteiro5.parte2) – Sobrecarga de Métodos;

1 – Crie o pacote roteiro5.parte2 com a cópia das classes Principal e Loja implementadas na parte1.

2 - Iremos agora aplicar os nossos conhecimentos de **Sobrecarga de Métodos**. A solução adotada no item 5 da Parte1 apesar de funcionar não é muito adequada. Podemos utilizar recursos da OO para tornar o código mais limpo e até reutilizável.

O adequado neste caso é aplicar a sobrecarga no método construtor. Ou seja, teremos 2 construtores que recebem diferentes parâmetros, cada um deve ser utilizado conforme a necessidade.

```
public class Loja {
    private String nomeFantasia;
    private String razaoSocial;
    private String cnpj;
    private double valorFat;
    private double area;

    public Loja(String nomeFantasia, String razaoSocial, String cnpj){
        this.nomeFantasia = nomeFantasia;
        this.razaoSocial = razaoSocial;
        this.cnpj = cnpj;
    }

    public Loja(String nomeFantasia, String cnpj){

        this.nomeFantasia = nomeFantasia;
        this.razaoSocial = nomeFantasia;
        this.cnpj = cnpj;
    }

    {
      Implementar Gets e Sets
    }

}
```

2 – Aplique agora as chamadas adequadas na classe Principal.

```
public class Principal {

    public static void main(String[] args) {

      Loja loja01 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");

      Loja loja02 = new Loja("Mercadão do Povo", "10101010");

    }
}
```

3 – Com sobrecarga do método construtor podemos otimizar o código ainda mais promovendo reutilização de código.

Podemos apenas chamar o 1º Construtor de dentro do 2º Construtor passando os parâmetros da forma desejada. Assim eliminamos linhas de código desnecessárias, pois estaremos de fato fazendo reutilização de código.

```
public class Loja {
    private String nomeFantasia;
    private String razaoSocial;
    private String cnpj;
    private double valorFat;
    private double area;

    public Loja(String nomeFantasia, String razaoSocial, String cnpj){
        this.nomeFantasia = nomeFantasia;
        this.razaoSocial = razaoSocial;
        this.cnpj = cnpj;
    }

    public Loja(String nomeFantasia, String cnpj){

        this(nomeFantasia, nomeFantasia, cnpj);

        this.nomeFantasia = nomeFantasia;
        this.razaoSocial = nomeFantasia;
        this.cnpj = cnpj;
    }

    {
      Implementar Gets e Sets
    }

}
```

4 – Para atividade final faça os testes abaixo na classe Principal. O objetivo é compreender como funciona os objetos instanciados e como testá-los.

Abaixo fizemos 4 testes (Teste1, Teste2, Teste3, Teste4, Teste5). Avalie os resultados apresentados e a diferença entre eles.

O que conseguiu concluir sobre os testes feitos ?
No Teste5 é possível utilizar o operador **.equals** ?

```
public class Principal {

    public static void main(String[] args) {

      Loja loja01 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");
      Loja loja02 = new Loja("Mercadão do Povo", "", "10101010");
      Loja loja03 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");

      loja01.setValorFat(10000);
      loja02.setValorFat(20000);
      loja03.setValorFat(10000);

      System.out.println(" ****** Teste 1 ********");
      if ( loja01.getNomeFantasia() == loja03.getNomeFantasia() ){
          System.out.println("Lojas Iguais");
      }
      else {
          System.out.println("Lojas Diferentes");
      }

      System.out.println(" ****** Teste 2 ********");
      if ( loja01.getNomeFantasia().equals(loja03.getNomeFantasia())  ){
          System.out.println("Lojas Iguais");
      }
      else {
          System.out.println("Lojas Diferentes");
      }

      System.out.println(" ****** Teste 3 ********");
      if ( loja01 == loja03  ){
          System.out.println("Lojas Iguais");
      }
      else {
          System.out.println("Lojas Diferentes");
      }

      System.out.println(" ****** Teste 4 ********");
      if ( loja01.equals(loja03)  ){
          System.out.println("Lojas Iguais");
      }
      else {
          System.out.println("Lojas Diferentes");
      }

      System.out.println(" ****** Teste 5 ********");
      if ( loja01.getValorFat() == loja03.getValorFat()  ){
          System.out.println("Faturamentos Iguais");
      }
      else {
          System.out.println("Faturamentos Diferentes");
      }

  }
}
```

Parte 2 (roteiro5.parte3) – Métodos Estáticos e Métodos não Estáticos

Os métodos static tem um relacionamento com uma classe como um todo, enquanto os métodos que não são static são associados a uma instância de classe específica (objeto) e podem manipular as variáveis de instância do objeto.

1 – Crie o pacote roteiro5.parte3 com a cópia das classes Principal e Loja implementadas na parte2.

2 – Crie um método na classe Loja para comparar se o faturamento de uma loja é superior ao de outra loja. Ou seja, este método deverá receber como parâmetro 2 lojas, e dentro deste método deveremos implementar a lógica para comparar as duas.

Esta solução pode ser implementada utilizando tanto um método estático quanto um não estático. O importante é entender como funciona do ponto de vista da OO, e principalmente quando e como utilizar.

```
public class Loja {
    private String nomeFantasia;
    private String razaoSocial;
    private String cnpj;
    private double valorFat;
    private double area;

    {
      Construtores
    }

    {
      Gets e Sets
    }


    public static void compararFat_static(Loja lojaA, Loja lojaB ){

        if (lojaA.getValorFat() > lojaB.getValorFat()){
            System.out.println("Loja de Maior Fat : " + lojaA.nomeFantasia);
        }
        else {
            System.out.println("Loja de Maior Fat : " + lojaB.nomeFantasia);
        }
    }

    public void compararFat_naoStatic(Loja lojaB ){

        if (this.getValorFat() > lojaB.getValorFat()){
            System.out.println("Loja de Maior Fat : " + this.nomeFantasia);
        }
        else {
            System.out.println("Loja de Maior Fat : " + lojaB.nomeFantasia);
        }
    }


}
```

3 – Como sugestão para teste, implemente o trecho de código abaixo na classe Principal

```
public class Principal {

    public static void main(String[] args) {

      Loja loja01 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");
      Loja loja02 = new Loja("Mercadão do Povo", "", "10101010");
      Loja loja03 = new Loja("Lojão da Cidade", "Lojão Comércio LTDA", "11223344");

      loja01.setValorFat(10000);
      loja02.setValorFat(20000);
      loja03.setValorFat(10000);


       * * *
       * * *
       * * *
       * * *

      System.out.println(" ****** Comparação com método estático ********");
      Loja.compararFat_static(loja01, loja02);

      System.out.println(" ****** Comparação com método NÃO estático ********");
      loja01.compararFat_naoStatic(loja02);

      System.out.println(" ****** Comparação com método NÃO estático ********");
      loja02.compararFat_naoStatic(loja01);


  }
}
```

4 – O que se espera é que nas 3 comparações acima seja apresentado o a mesma loja como a de maior faturamento.

O que conseguiu entender sobre a utilização dos dois métodos (Estáticos e Não Estáticos) ?

5 – Como exercício, faça com que os dois métodos (Estáticos e Não Estáticos) retornem a loja de maior faturamento. Assim, a impressão de quem é maior deve ser feita na classe Principal e não na classe Loja.