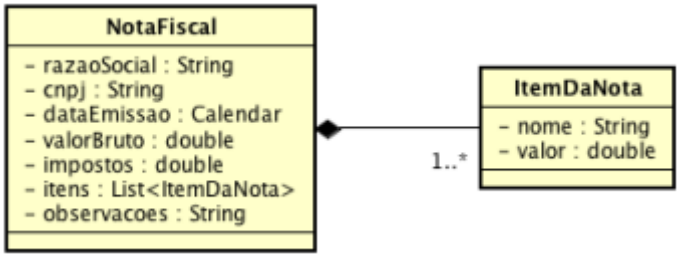
Universidade de Mogi das Cruzes

Padrões de Projetos Orientados a Objetos

Padrão de Criação Builder

Vamos propor um exemplo de criação de um objeto que represente uma Nota Fiscal, cujos dados necessários sejam, Razão Social, CNPJ, Data de emissão, Valor Bruto, Impostos, Itens da Nota e Observações.



O código desta classe pode ser definido da seguinte forma:

```
15  public class NotaFiscal {
16
17      private String razaoSocial;
18      private String cnpj;
19      private Calendar dataEmissao;
20      private double valorBruto;
21      private double impostos;
22      private List<ItemDaNota>itens;
23      private String observacoes;
24
25      public NotaFiscal(String razaoSocial, String cnpj, Calendar dataEmissao,
26              double valorBruto, double impostos, List<ItemDaNota> itens, String observacoes) {
27          this.razaoSocial = razaoSocial;
28          this.cnpj = cnpj;
29          this.dataEmissao = dataEmissao;
30          this.valorBruto = valorBruto;
31          this.impostos = impostos;
32          this.itens = itens;
33          this.observacoes = observacoes;
34      }
35
36
37      //sets e gets
38
39  }
```

A classe ItemDaNota

```java
public class ItemDaNota {
    private String nome;
    private double valor;

    public ItemDaNota(String nome, double valor) {
        this.nome = nome;
        this.valor = valor;
    }

  //gets e sets
}
```

Uma classe de teste para Nota Fiscal

```java
public class TesteNotaFiscal {

    public static void main(String[] args) {

        List<ItemDaNota> itens = Arrays.asList(new ItemDaNota("item 1", 200.0),
                                               new ItemDaNota("item 2", 400.0));

        double total=0;
        for(ItemDaNota item: itens){
            total += item.getValor();
        }


        double impostos = total * 0.5;

        NotaFiscal nf = new NotaFiscal("razao social", "cnpf", Calendar.getInstance(),
                                       total, impostos, itens, "obs qualquer");


    }
}
```

Veja que o construtor dessa classe é extenso e difícil de entender. Além disto, a regra de criação do objeto acabou ficando espelhado pelo código do método main. Pode acontecer de surgir a necessidade de paramentos opcionais. Nesse caso, o programador começa a criar diferentes construtores, cada um com uma possível combinação de parâmetros de entrada, tornando a sua classe mais difícil ainda de ser lida.

Para resolver o problema, separaremos a lógica da criação desse objeto complexo na classe NotaFiscalBuilder. Essa nova classe será responsável por pedir todos os parâmetros necessários, montar o que precisa, e enfim produzir uma Nota Fiscal, de forma com que o cliente dessa classe consiga entender o que está acontecendo.

Veja o código abaixo:

```
11  public class TesteCriador {
12      public static void main(String[] args) {
13
14          NotaFiscalBuilder builder = new NotaFiscalBuilder();
15          builder.paraEmpresa("Minha Empresa");
16          builder.comCnpj("4561236");
17          builder.comItem(new ItemDaNota("item 1", 100.0));
18          builder.comItem(new ItemDaNota("item 2", 200.0));
19          builder.comItem(new ItemDaNota("item 3", 300.0));
20          builder.comObservacoes("entregar nf pessoalmente");
21          builder.naDataAtul();
22
23          NotaFiscal nf= builder.constroi();
24
25      }
26  }
```

Pede-se: Implementar o código da ***NotaFiscalBuilder*** conforme a aplicação acima.

Desafio 1: altere a classe ***NotaFiscalBuilder*** para utilizar-se da notação de interface fluente. (Fazer um projeto separado).

Desafio 2: altere a classe NotaFiscal para incluir o builder como classe interna dela. (Fazer um projeto separado).