Escrever código "feio" é natural; todo desenvolvedor faz isso uma vez ou outra. Não porque lhe falta experiência ou conhecimento, mas muitas vezes ele está tão focado no algoritmo que está implementando, no problema que está sendo resolvido, que acaba por não pensar bem na qualidade do código que escreveu.

Quantas vezes você não voltou a um código seu, e viu que tinha dado um péssimo nome para a variável, ou que tinha escrito um método com mais linhas do que deveria? Nesses casos, precisamos melhorar nosso código. Quando melhoramos nosso código, seja renomeando uma variável ou quebrando uma classe grande em várias classes pequenas, damos o nome de **refatoração**.

Ao longo desse curso, estudaremos diversas maneiras de melhorar/refatorar o nosso código. Algumas dessas técnicas são bem simples; outras são mais avançadas. É importante que o desenvolvedor tenha em seu cinto de práticas todas essas técnicas para melhorar seu código constantemente.

Por exemplo, um erro comum nosso ao longo do dia de desenvolvimento é a criação de classes maiores do que deveriam. Veja, por exemplo, a classe `GeracaoDeNotaFiscal` abaixo:

```
public class GeradorDeNotaFiscal {

    public NotaFiscal gera(Fatura fatura) {

        // calcula valor do imposto
        double valor = fatura.getValorMensal();
        double imposto = 0;
        if(valor < 200) {
            imposto = valor * 0.03;
        }
        else if(valor > 200 && valor <= 1000) {
            imposto = valor * 0.06;
        }
        else {
            imposto = valor * 0.07;
        }

        NotaFiscal nf = new NotaFiscal(valor, imposto);

        // envia email
        String msgDoEmail = "Caro cliente,<br/>";
        msgDoEmail += "É com prazer que lhe avisamos que sua nota fiscal
                + "gerada no valor de " + nf.getValorLiquido() + ".<br/>"
        msgDoEmail += "Acesse o site da prefeitura e faça o download.<br/
        msgDoEmail += "Obrigado!";

        System.out.println(msgDoEmail);

        // salva no banco
        String sql = "insert into notafiscal (cliente, valor)"+
                     "values (?," + nf.getValorLiquido() + ")";

        System.out.println("Salvando no banco" + sql);

        return nf;
    }
}
```

Tente entender o que ela faz: Ela calcula o valor do imposto da nota fiscal, envia um e-mail (com uma mensagem na tela, mas imagine que ali tenhamos o código de envio de e-mail) e salva no banco. Ela é bem extensa, e difícil de ser entendida rapidamente.

Só o método `gera()` tem cerca de 30 linhas. Imagine agora casos piores: métodos com 100, 200 ou 300 linhas. Quanto mais linha de código, mais difícil é de entendê-lo. Uma primeira refatoração possível para esse tipo de código é dividir esse método público em vários métodos privados.

Repare que temos 3 coisas diferentes acontecendo nesse código: a geração da NF, o envio do e-mail, e a persistência da nota fiscal no banco. Os comentários até nos indicam isso. Podemos colocar cada um desses em um método privado:

```
public class GeradorDeNotaFiscal {

    public NotaFiscal gera(Fatura fatura) {

        NotaFiscal nf = geraNf(fatura);

        enviaEmail(nf);
        salvaNoBanco(nf);

        return nf;
    }

    private void salvaNoBanco(NotaFiscal nf) {
        // salva no banco
        String sql = "insert into notafiscal (cliente, valor)"+
                     "values (?," + nf.getValorLiquido() + ")";

        System.out.println("Salvando no banco" + sql);
    }

    private void enviaEmail(NotaFiscal nf) {
        // envia email
        String msgDoEmail = "Caro cliente,<br/>";
        msgDoEmail += "É com prazer que lhe avisamos que sua nota fiscal
                + "gerada no valor de " + nf.getValorLiquido() + ".<br/>"
        msgDoEmail += "Acesse o site da prefeitura e faça o download.<br/
        msgDoEmail += "Obrigado!";

        System.out.println(msgDoEmail);
    }

    private NotaFiscal geraNf(Fatura fatura) {
        // calcula valor do imposto
        double valor = fatura.getValorMensal();
        double imposto = 0;
        if(valor < 200) {
            imposto = valor * 0.03;
        }
        else if(valor > 200 && valor <= 1000) {
            imposto = valor * 0.06;
        }
        else {
            imposto = valor * 0.07;
        }

        NotaFiscal nf = new NotaFiscal(valor, imposto);
        return nf;
    }
}
```

Veja que a classe continua grande, mas agora já é mais fácil de ler. Veja que o método público tem agora 3 linhas. Uma delas gera a nota fiscal, outra envia o e-mail, outra salva no banco. Agora, se você não quiser ler o código do envio de e-mail, você pode simplesmente ignorar o método privado! Isso não acontecia antes. Se aparecer um bug na hora de salvar o dado no banco, você já sabe que o erro está dentro do método `salvaNoBanco()`; muito mais rápido e fácil.

Dividir um método público em métodos menores privados, é uma excelente técnica para melhorarmos um código. Isso, sem dúvida alguma, faz com que o desenvolvedor gaste menos tempo para entender o código.

Uma outra técnica que pode bem ser utilizada é a divisão dessas responsabilidades em classes diferentes. Ao invés de levarmos o trecho de código para um método privado, levamos para uma nova classe. Podemos criar uma classe que faz a consulta ao banco de dados, e uma outra que dispara o e-mail. E, na classe `GeradorDeNotaFiscal`, consumiremos essas novas classes.

Veja a refatoração feita abaixo:

```
public class EnviadorDeEmail {

    public void enviaEmail(NotaFiscal nf) {
        // envia email
        String msgDoEmail = "Caro cliente,<br/>";
        msgDoEmail += "É com prazer que lhe avisamos que sua nota fiscal
                + "gerada no valor de " + nf.getValorLiquido() + ".<br/>"
        msgDoEmail += "Acesse o site da prefeitura e faça o download.<br/
        msgDoEmail += "Obrigado!";

        System.out.println(msgDoEmail);
    }
}

public class NFDao {
    public void salvaNoBanco(NotaFiscal nf) {
        // salva no banco
        String sql = "insert into notafiscal (cliente, valor)"+
                     "values (?," + nf.getValorLiquido() + ")";

        System.out.println("Salvando no banco" + sql);
    }

}

public class GeradorDeNotaFiscal {

    public NotaFiscal gera(Fatura fatura) {

        NotaFiscal nf = geraNf(fatura);

        new EnviadorDeEmail().enviaEmail(nf);
        new NFDao().salvaNoBanco(nf);

        return nf;
    }

    private NotaFiscal geraNf(Fatura fatura) {
        // metodo continua aqui
    }
}
```

Agora veja que temos 3 classes, pequenas, cada um com a sua responsabilidade específica. Mais pra frente discutiremos mais sobre acoplamento e coesão, mas por enquanto, veja que cada classe tem um código curto e fácil de ser entendido.

Uma boa dica para saber se o método está grande ou não, é se ele cabe inteiro na tela do seu computador. Se você precisa rolar a tela para ver o método todo, talvez valha a pena dividí-lo. Como dividir? Em métodos privados ou classes, como acabamos de mostrar.