



Desenvolvimento de Sistemas

Refatoração de código: técnicas, boas práticas, *code smells*

A refatoração é o processo de reestruturação do código, sem alterar a funcionalidade original deste.

Por exemplo, para você entender melhor, suponha que existe um método que compara dois números e retorna **true**, se o primeiro número for maior, e **false** se for menor:

```
public boolean max(int a, int b) {  
    if ( a > b) {  
        return true;  
    } else if ( a == b) {  
        return false;  
    } else {  
        return false;  
    }  
}
```

Esse código, apesar de executar o que é necessário, é bastante inadequado. Nesse caso em específico, para uma refatoração, pode-se analisar a necessidade dos blocos *if-else*. Se se pensar um pouco mais, pode ser escrito um método de seis linhas de maneira mais concisa:


```
public boolean max(int a, int b) {  
    return a > b;  
}
```

Agora, sim, tem-se um método simples e elegante que realiza a mesma operação do exemplo. É assim que a refatoração funciona: **você altera a estrutura do código sem afetar a essência dele**. Existem muitos métodos e técnicas de refatoração, os quais serão examinados mais detalhadamente no decorrer deste texto.

O que é refatoração?

A refatoração é um processo sistemático de alterar o código-fonte previamente construído, sem introduzir novas funcionalidades ou alterar o funcionamento fundamental do *software* em questão. Você pode pensar nela como um pequeno ajuste de programação destinado a otimizar a estrutura, a definição e a implementação do código subjacente sem interferir na funcionalidade original do *software*.

Os resultados finais são geralmente melhorias na legibilidade, na manutenção e na extensibilidade do *software*. Algumas das ações que as equipes de desenvolvimento de *software* geralmente realizam durante a refatoração são:

- 
- ◆ Reduzir o tamanho do código
 - ◆ Reestruturar o código confuso em um código mais simples
 - ◆ Limpar o código para torná-lo mais organizado
 - ◆ Remover do código comentários redundantes e não utilizados
 - ◆ Eliminar repetições desnecessárias
 - ◆ Criar um código reutilizável
 - ◆ Quebrar funções longas em funções mais simples e gerenciáveis

Então, todo processo de refatoração normalmente compreende várias “microrrefatorações”, as quais são executadas em pequenas partes para minimizar os riscos no sistema à medida que o código está sendo reestruturado. Isso mantém o *software* funcionando perfeitamente, sem grandes interrupções de serviço que possam comprometer as operações de negócios.

Vale a pena notar que, enquanto as instâncias individuais de microrrefatorações realizam pequenas transformações, as sequências padronizadas acabam se combinando para realizar uma extensa reestruturação.

No entanto, não espere seguir um cronograma regular, pois não há diretrizes de tempo para refatoração. O processo nem aparece como uma tarefa agendada nos planos de desenvolvimento de software.

A refatoração deve fazer parte da **programação diária** do desenvolvedor. Caso contrário, você deve pelo menos executá-la sempre que pretender introduzir atualizações ou novos recursos no código existente.

Por exemplo, você pode querer revisar seu código-fonte antes de inserir novos recursos no *software*. É aqui que você avalia a estrutura do código para sinalizar todas as “anomalias” que podem comprometer a integração perfeita de novos recursos. Assim, caso descubra algum ponto de refatoração, você deve resolver os problemas sem interferir nas principais funcionalidades do *software*.

Lembre-se, porém, de que você não precisa se restringir a um processo totalmente manual. Muitos ambientes de desenvolvimento de aplicativos agora oferecem alguma forma de suporte para refatoração automatizada, com o objetivo de facilitar a execução dos aspectos mecânicos.

Qualquer que seja a abordagem escolhida para o código do *software*, a refatoração deve sempre ter precedência sobre a adição de novos recursos. Quanto mais frequentemente você conseguir executar os procedimentos, mais profundamente poderá otimizar seu código para escalabilidade. Isso significa que você terá menos *bugs* para lidar ao tentar expandir os recursos e as funcionalidades.

Caso contrário, desconsiderar a refatoração pode levar ao acúmulo gradual de dívida técnica ou dívida de código. Esse acúmulo, em particular, ocorre quando as equipes de desenvolvimento de *software* optam por priorizar atualizações de funcionalidade em vez de refatoração. Assim, à medida que novas funcionalidades chegam, pontos fracos não resolvidos no código-fonte desencadeiam todos os tipos de problemas técnicos, que se acumulam ao longo do tempo e se transformam em grandes complexidades.

Benefícios da refatoração

Independentemente das técnicas de refatoração que você for aplicar, todo esse processo de alterar sistematicamente o código-fonte subjacente oferece os seguintes benefícios:

Melhora da legibilidade

Mesmo programadores experientes acham difícil ler e interpretar o código em sistemas que ainda não foram refatorados. Códigos que não sofreram refatoração geralmente são carregados de repetições, funções desnecessariamente longas, linhas redundantes, fluxos incoerentes e problemas de sintaxe.

Veja o exemplo a seguir, que demonstra, em um mesmo método, todo o cálculo para obter o percentual de um valor:

```
public Double reajuteSalPorPorcentagem(Double percentual){
    Double reajuste,
    Double novoSalario;
    reajuste = (this.salario * percentual) / 100;
    novoSalario = this.salario + reajuste;
    return novoSalario;
}
```

Separando o método exemplificado em dois momentos, é possível melhorar a legibilidade:

```
public Double reajuteSalPorPorcentagem(Double percentual){
    return this.salario + encontrarValorPorc(Double percentual);
}

public Double encontrarValorPorc(Double percentual){
    return (this.salario * percentual) / 100;
}
```

Mesmo que o método **encontrarValorPorc()** seja dificilmente reutilizado, o benefício é que o código fica mais legível.

A refatoração, portanto, é extremamente útil. O processo ajuda a identificar as inconsistências e, em seguida, organizar o código para tornar tudo muito mais simples para todos.

Melhora do desempenho

Além de simplificar o código-fonte, a refatoração elimina progressivamente todos os tipos de anomalias básicas. Isso evita o acúmulo de dívida técnica, o que comprometeria substancialmente as funcionalidades e os recursos do aplicativo. Nesse sentido, as técnicas de refatoração protegem e melhoram o desempenho do *software* a longo prazo.

Economia de dinheiro e de tempo

Ao diminuir a necessidade de manutenção e aumentar a legibilidade e o desempenho, o código refatorado permite economizar dinheiro e tempo. Mais especificamente, ele otimiza o código subjacente para simplificar as tarefas de manutenção, minimizar o tamanho do sistema, eliminar dívidas técnicas potencialmente dispendiosas e aumentar a eficiência geral do aplicativo. Isso, com o tempo, reduz até a quantidade correspondente de recursos consumidos – o que se traduz em mais dinheiro disponível.

Técnicas e boas práticas para utilizar na refatoração

Agora que você sabe o que é refatoração de código e conhece muitos dos benefícios que ela proporciona, como exatamente você refatora o código?

Existem muitas práticas diferentes de refatoração, mas aqui estão algumas das mais comuns. Confira a seguir as cinco principais técnicas de refatoração de código na linguagem Java:

Ferramentas de refatoração no NetBeans 13

O IDE NetBeans conta com uma ferramenta de refatoração própria. Uma das operações de refatoração mais utilizadas é a renomeação de uma palavra-chave ou de um identificador específico. Por exemplo, se você deseja alterar o nome da classe que foi criada, precisa alterar o nome da classe, bem como o arquivo **.java**. Com a refatoração do NetBeans, a alteração em uma instância é refletida automaticamente nas outras instâncias.

Para usar a refatoração para renomear, selecione o nome da classe que deseja alterar, clique com o botão direito do *mouse* e navegue até **Refactor > Rename**. Uma pequena caixa de diálogo é aberta solicitando o novo nome. Digite o novo nome e observe a mudança que ocorre:

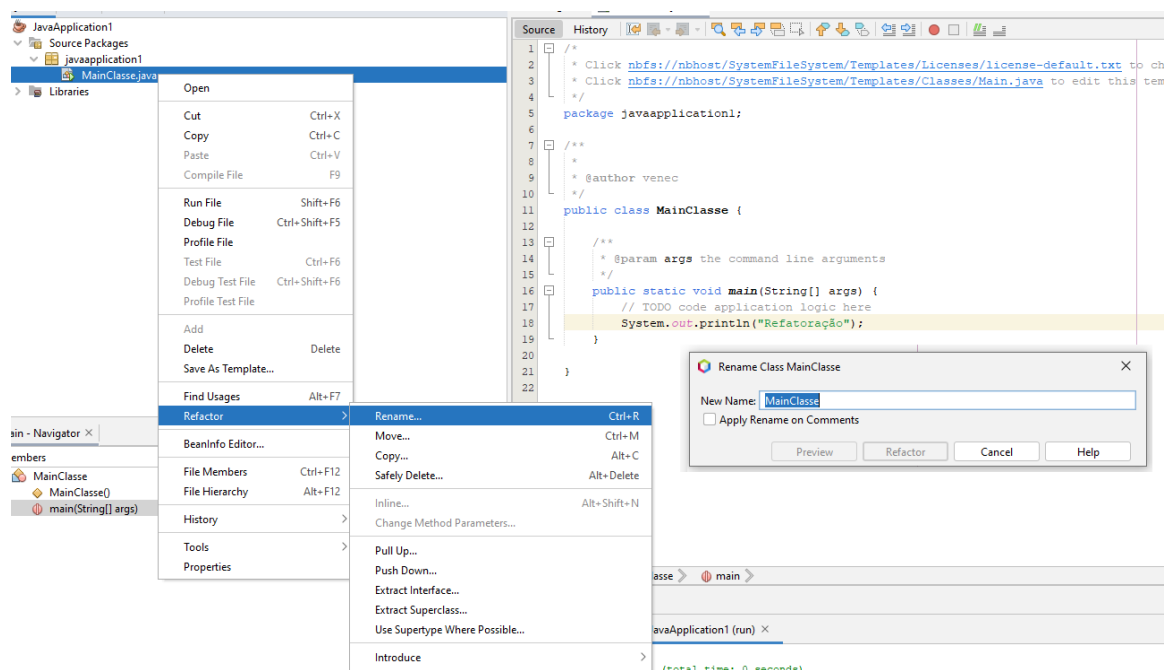



Figura 1 – Opção **Rename** no NetBeans

 Fonte: Senac EAD (2022)

Ao informar um novo nome, a classe e o arquivo serão alterados, mas não apenas eles. Caso haja algum objeto inicializado como essa classe, o código de inicialização também será atualizado imediatamente.

Existem várias operações de refatoração, cada uma com o próprio significado, por exemplo: mover e excluir com segurança, encontrar todas as instâncias do uso da classe e alterá-las de acordo com a operação concluída. A exclusão segura ajuda a identificar os possíveis locais onde os erros podem ocorrer.

Code smells

Uma coisa que a maioria dos desenvolvedores eventualmente encontra ao trabalhar com sistemas complexos ou em grandes equipes são os *code smells* (cheiros de código). *Code smells* são o resultado de uma programação ruim ou mal orientada. Essas situações no código podem ser rastreadas diretamente em erros cometidos pelo programador durante o processo de codificação.

Um código particularmente “fedorento” pode ser ineficiente, complexo e difícil de alterar e manter.

Embora os *code smells* nem sempre indiquem um problema particularmente sério, segui-los geralmente leva a descobertas de diminuição da qualidade do código, esgotamento de recursos do *software* ou até vulnerabilidades críticas de segurança incorporadas ao código. No mínimo, as equipes devem realizar alguns testes aprofundados no código, os quais comumente revelam algumas áreas críticas do código que precisam de reparos.

Você verá agora como as equipes de *software* podem identificar “cheiros de código” e usá-los para manter um grau mais alto de limpeza de código. Também examinará como técnicas, tais como a refatoração de código, desempenham papéis essenciais ao lidarem com *code smells* e corrigirem os problemas.

O que causa *code smells*

Normalmente, os *code smells* resultam de uma falha em escrever o código de acordo com os padrões necessários. Em outros casos, pode ser que a documentação necessária para definir claramente os padrões e as expectativas de desenvolvimento do projeto estava incompleta, imprecisa ou nem mesmo existia.

Há muitas situações que podem causar “cheiros de código”, como dependências impróprias entre módulos, atribuição incorreta de métodos a classes ou duplicação desnecessária de segmentos de código. O código que é particularmente “malcheiroso” pode eventualmente causar problemas profundos de desempenho e dificultar a manutenção de *softwares*.

No entanto, um *code smell* não é um *bug* real – é provável que o código ainda seja compilado e funcione conforme o esperado. Os *code smells* são simplesmente indicações de possíveis violações da disciplina de código e dos princípios de *design*. Assim, é possível que a fonte de um *code smell* possa causar problemas em cascata e falhas ao longo do tempo. Ela também é um bom indicador de que um esforço de fatoração de código será preciso.

Eliminando *code smells* com refatoração

A refatoração de código é uma das maneiras mais eficazes de eliminar *code smells* e manter uma boa higiene de código. O melhor momento para refatorar o código é antes de adicionar atualizações ou novos recursos a um *software*, pois é uma boa prática limpar o código existente antes que os programadores adicionem qualquer novo código.

Outro bom momento para refatorar o código é depois que uma equipe implantou o código na produção. Afinal, os desenvolvedores teriam mais tempo do que o normal para limpar o código antes de receberem uma nova tarefa ou um novo projeto.

Uma ressalva para a refatoração é que as equipes devem se certificar de que há uma cobertura completa do teste antes de refatorar o código de um aplicativo. Caso contrário, o processo de refatoração pode simplesmente reestruturar partes quebradas do aplicativo sem nenhum ganho. A refatoração regular não é uma boa ideia ao enfrentar cronogramas de lançamento apertados – os testes necessários levam muito tempo e podem impedir que a equipe libere o aplicativo no prazo.

A refatoração engloba várias práticas específicas de higiene de código. Porém, quando se trata de eliminar “maus cheiros” no código, existem três técnicas particularmente eficazes: uma que se concentra em métodos, outra que se concentra nas chamadas de método e outra que se concentra em classes.

Desafio 1

Considere esta classe:

```
public class A{

    public void realizaOperacao()
    {
        int x = 10;

        int y = 0;
        y = obterValor() * 2;
        System.out.println(y);

        int z = y + 1;
        //outros códigos
    }

    public int obterValor()
    {
        java.util.Random gerador = new java.util.Random();
        return gerador.nextInt();
    }
}
```

Extraia o trecho destacado em azul para um novo método chamado de **calculaValor**.

Desafio 2

Analise esta classe e realize as refatorações que identificar necessárias:

```
public class Aluno {  
    public String nome;  
    public float[] notas;  
    public String rua;  
    public int numero;  
    public String cidade;  
  
    public void ajustaEndereco(String rua, int numero, String cidade)  
    {  
        this.rua = rua;  
        this.numero = numero;  
        this.cidade = cidade;  
    }  
  
    public float calculaMedia()  
    {  
        float media = 0;  
        for(float n : notas)  
            media += n;  
        media = media / notas.length;  
  
        if(media >= 7)  
            System.out.println("Aprovado");  
        else  
            System.out.println("Reprovado");  
  
        return media;  
    }  
}
```

Encerramento

A refatoração deve ser uma atividade recorrente, e é fundamental que seja um esforço colaborativo. Ao realizar regularmente tarefas de higiene de código em conjunto, as equipes de *software* podem criar estratégias centralizadas para identificar “maus cheiros” de código e aprender com os erros.

Também é importante que os testadores aprendam o processo de reestruturação de código, para que consigam melhorar os casos e os procedimentos de teste existentes, escrever melhor o código de automação de teste e reduzir as tarefas de manutenção manual.

Nunca esqueça que é fundamental garantir que o código seja testável antes do início da refatoração. Isso inclui – mas não se limita a – testes de unidade, testes de regressão e testes de integração. Para tanto, também é importante envolver a equipe de teste em todo o processo de refatoração, pois os testes podem falhar quando as equipes de desenvolvimento começarem a alterar o código.