REFATORAÇÃO

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

26 de fevereiro de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

Refatoração (refactoring)

- Uma *(pequena)* modificação no sistema que não altera o seu comportamento funcional,
- Mas que melhora alguma qualidade não funcional:
  - simplicidade
  - flexibilidade
  - clareza
  - desempenho

refatoração (s. f.): uma mudança na estrutura interna de um software para torná-lo mais compreensível e simples de ser modificado, sem que seu comportamento seja alterado.

refatorar (v. tr.): reestruturar um software aplicando uma série de refatorações, sem que seu comportamento seja alterado.

EXEMPLOS DE REFATORAÇÃO

- Mudança no nome de variáveis
- Mudanças nas interfaces dos objetos
- Pequenas mudanças arquiteturais
- Encapsular código repetido em um novo método
- Generalização de métodos:
  `float raizQuadrada(float x)` $\Rightarrow$
  `float raiz(float x, int n)`

QUANDO REFATORAR?

Sempre há duas possibilidades:

1. Melhorar o código existente
2. Jogar tudo fora e começar do zero

É sua responsabilidade

avaliar a situação e optar por um ou por outro.

APLICAÇÕES

1. Melhorar código antigo e/ou feito por outros programadores
2. Desenvolvimento incremental à la TDD

- Em geral um passo de refatoração é tão simples que parece que ele não vai ajudar muito
- Mas quando se juntam 50 passos bem escolhidos e em sequência, o código melhora radicalmente

PASSOS DE REFATORAÇÃO

- Cada passo é trivial
- Demora alguns segundos ou poucos minutos para ser realizado
- Deve ser feito sistematicamente
- O segredo é ter um bom vocabulário de refatorações e saber quando aplicá-las

PASSOS DE REFATORAÇÃO

- Cada passo é trivial
- Demora alguns segundos ou poucos minutos para ser realizado
- Deve ser feito sistematicamente
- O segredo é ter um bom vocabulário de refatorações e saber quando aplicá-las

Vocês já refatoraram código!

- Refatoração sempre existiu, só não tinha um nome
- Mas agora nós temos um vocabulário comum e um catálogo com as refatorações mais utilizadas

ORIGENS

- Surgiu na comunidade de Smalltalk nos anos 80/90
- Desenvolveu-se formalmente no grupo de pesquisa do Prof. Ralph Johnson (Universidade de Illinois)
  - Tese de doutorado de William Opdyke (1992)
  - John Brant e Don Roberts: *The Refactoring Browser Tool*
- Kent Beck (XP) na indústria

ESTADO ATUAL

- Hoje em dia é um dos preceitos de métodos ágeis
- Não é limitado a Smalltalk
- Pode ser usado em qualquer linguagem (orientada a objetos ou não)

CATÁLOGO DE REFATORAÇÕES

- O catálogo online de Fowler tem 91 refatorações
- Análogo aos padrões de desenho orientado a objetos (que veremos mais pra frente)
- Vale a pena gastar algumas horas com o livro e o site de Fowler.

DICA #1

Quando você tiver que adicionar uma funcionalidade a um programa e o código do programa não estiver estruturado de uma forma que torne a implementação desta funcionalidade conveniente, primeiro refatore de modo a facilitar a implementação da funcionalidade e, só depois, implemente-a.

O PRIMEIRO PASSO EM QUALQUER REFATORAÇÃO

- Antes de começar, verifique se você tem um conjunto sólido de testes automáticos para verificar a funcionalidade do código a ser refatorado
- Refatorações podem adicionar erros
- Os testes ajudarão a detectar esses erros assim que ele forem introduzidos

EXTRACT METHOD

Contexto: você tem um fragmento de código que pode ser agrupado

Resumo: transforme um fragmento de código em um método com um nome que explica o propósito do código

```
void printOwing() {
  printBanner();

  //print details
  System.out.println("name:  "
         + _name);
  System.out.println("amount "
         + getOutstanding());
}
```

EXTRACT METHOD

Contexto: você tem um fragmento de código que pode ser agrupado

Resumo: transforme um fragmento de código em um método com um nome que explica o propósito do código

```
void printOwing() {
  printBanner();

  //print details
  System.out.println("name:  "
         + _name);
  System.out.println("amount "
         + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails (double outstanding) {
  System.out.println ("name:  " + _name);
  System.out.println ("amount " + outstanding);
}
```

REPLACE TEMP WITH QUERY

Contexto: você está usando uma variável temporária para guardar o resultado de uma expressão

Resumo: extraia a expressão como um método. Substitua todas as referências para a variável temporária pela expressão. O novo método poderá então ser usado por outros métodos.

```
double basePrice = _quantity * _itemPrice;
if (basePrice > 1000)
  return basePrice * 0.95;
else
  return basePrice * 0.98;
```

REPLACE TEMP WITH QUERY

Contexto: você está usando uma variável temporária para guardar o resultado de uma expressão

Resumo: extraia a expressão como um método. Substitua todas as referências para a variável temporária pela expressão. O novo método poderá então ser usado por outros métodos.

```
double basePrice = _quantity * _itemPrice;
if (basePrice > 1000)
  return basePrice * 0.95;
else
  return basePrice * 0.98;
```

```
  if (basePrice() > 1000)
  return basePrice() * 0.95;
else
  return basePrice() * 0.98;
...
double basePrice() {
  return _quantity * _itemPrice;
}
```

INLINE METHOD (117)

Contexto: a implementação de um método é tão clara quanto o seu nome.

Resumo: substitua a chamada ao método pela sua implementação. É bom para eliminar indireção desnecessária. Se você tem um grupo de métodos mal organizados, aplique *Inline Method* em todos eles seguido de uns bons *Extract Method*

```
int bandeiradaDoTaxi(int hora) {
  return(depoisDas22Horas(hora)) ? 2 : 1);
}
int depoisDas22Horas(int hora) {
  return hora > 22;
}
```

INLINE METHOD (117)

Contexto: a implementação de um método é tão clara quanto o seu nome.

Resumo: substitua a chamada ao método pela sua implementação. É bom para eliminar indireção desnecessária. Se você tem um grupo de métodos mal organizados, aplique *Inline Method* em todos eles seguido de uns bons *Extract Method*

```
int bandeiradaDoTaxi(int hora) {
  return(depoisDas22Horas(hora)) ? 2 : 1);
}
int depoisDas22Horas(int hora) {
  return hora > 22;
}
```

```
int bandeiradaDoTaxi (int hora) {
  return (hora > 22) ? 2 : 1);
}
```

INTRODUCE NULL OBJECT (260)

Contexto: você tem repetidos `if`s para verificar se uma variável é `null` ou não

Resumo: troque os valores `null` por um objeto null

```
if(cliente == null)
  plano = PlanoDeCobrança.básico();
else
  plano = cliente.plano();
```

INTRODUCE NULL OBJECT (260)

Contexto: você tem repetidos `if`s para verificar se uma variável é `null` ou não

Resumo: troque os valores `null` por um objeto null

```
if(cliente == null)
  plano = PlanoDeCobrança.básico();
else
  plano = cliente.plano();
```

```
class ClienteNull extends Cliente {
  boolean éNull() { return true; }
  PlanoDeCobrança plano () {
    return new PlanoDeCobrançaNull();
  }
}

// Substitui valores null por objeto null
cliente = (pedido.cliente != null) ? pedido.cliente : new ClienteNull();

// Use o objeto nulo como se fosse um objeto qualquer
plano = cliente.plano()
```

Quando o código cheira mal, refatore-o!

Cheiro	Refatoração a ser aplicada
Código duplicado	`Extract Method` (110) `Substitute Algorithm` (139)
Método muito longo	`Extract Method` (110) `Replace Temp With Query` (120) `Introduce Parameter Object` (295)
Classe muito grande	`Extract Class` (149) `Extract Subclass` (330) `Extract Interface` (341) `Duplicate Observed Data` (189)

Quando o código cheira mal, refatore-o!

Cheiro	Refatoração a ser aplicada
Intimidade inapropriada	`Move Method` (142) `Move Field` (146) `Replace Inheritance with Delegation` (352)
Comentários	`Extract Method` (110) `Introduce Assertion` (267)
Muitos parâmetros	`Replace Parameter with Method` (292) `Preserve Whole Object` (288) `Introduce Parameter Object` (295)

OBSERVAÇÕES:

- Qualquer um consegue escrever código que o computador entende. Bons programadores escrevem código que pessoas conseguem entender
- Quando você sente que é preciso escrever um comentário para explicar melhor o código, tente refatorar primeiro
- Testes tem que ser automáticos e ser capazes de se autoverificarem

BIBLIOGRAFIA

- Material de curso dos profs. Fabio Kon e Alfredo Goldman `https://www.ime.usp.br/~kon/presentations/refatoracao2006.ppt`
- Catálogo de técnicas de refatoração de Martin Fowler `http://www.refactoring.com/catalog/`
- livro **Refactoring: Improving the Design of Existing Code** de Martin Fowler (com Kent Beck, John Brant, William Opdyke e Don Roberts)