PÓS-GRADUAÇÃO ALFA

FACULDADES ALFA ALVES ARIA

A MELHOR ESCOLA DE NEGÓCIOS DO CENTRO-OESTE

Diego Américo Guedes

Refatoração e Evolução de Software

GRUPO JOSÉ ALVES

Apresentação

- Bacharel em Ciências da Computação, UFG (2010)
- Mestrado em Ciências da Computação, UFG (2013)
- Campeão das regionais da Maratona de Programação em 2008 e 2009
- Trabalhei em projetos regionais, nacionais e mundiais de pesquisa na área de Ciências da Computação
- Fui professor Adjunto na Faculdade Senac por 2 anos (2015-2016)
- P&D goGeo (2014)
- Atualmente, Professor Substituto da UFG e Analista de Sistemas na Saneago

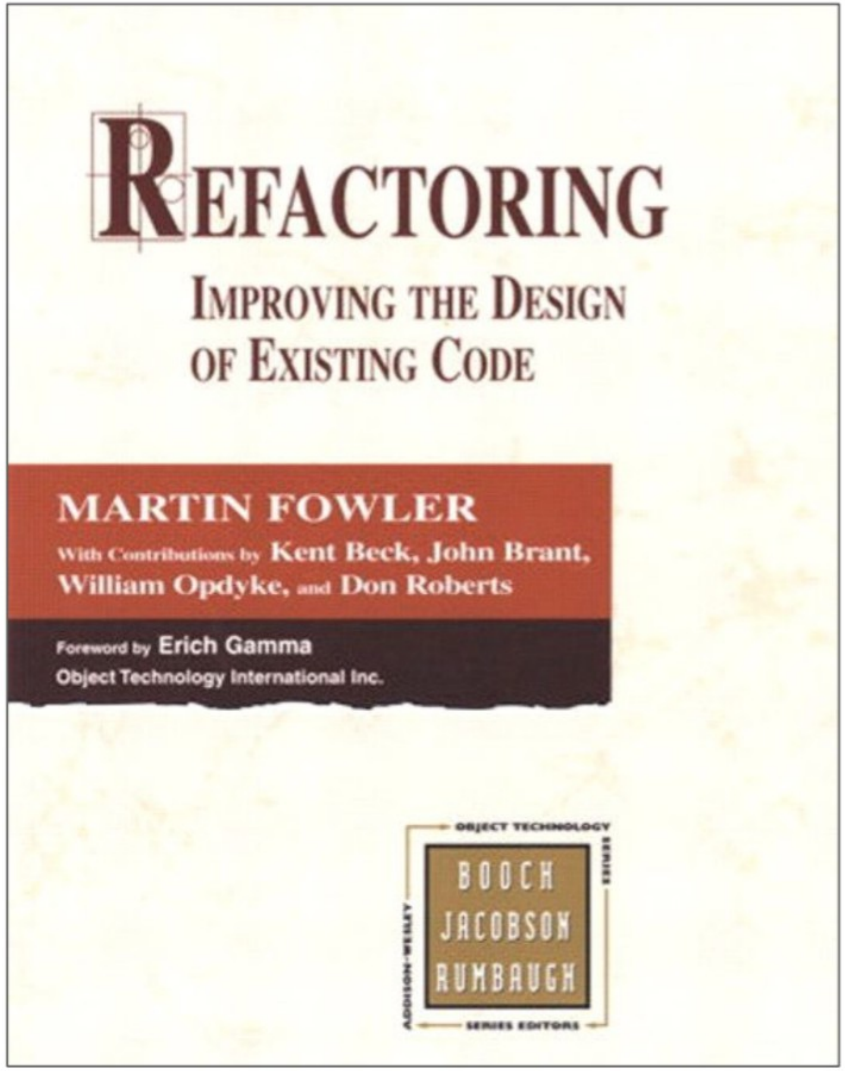
Exemplo de código

```
void imprimeDivida() {
// imprime cabeçalho
System.out.println("*****************************");
System.out.println("*** Dívidas do Cliente ****");
System.out.println("*****************************");
// calcula dívidas
double divida = 0.0;
int i = 0;
ArrayList<Itens> e = pedidos.getPedidos();
while (i < e.size()) {
Itens cada = (Itens) e.get(i);
divida += cada.getValor();
}
// imprime detalhes
System.out.println("nome: " + nome);
System.out.println("divida total: " + divida);
}
```

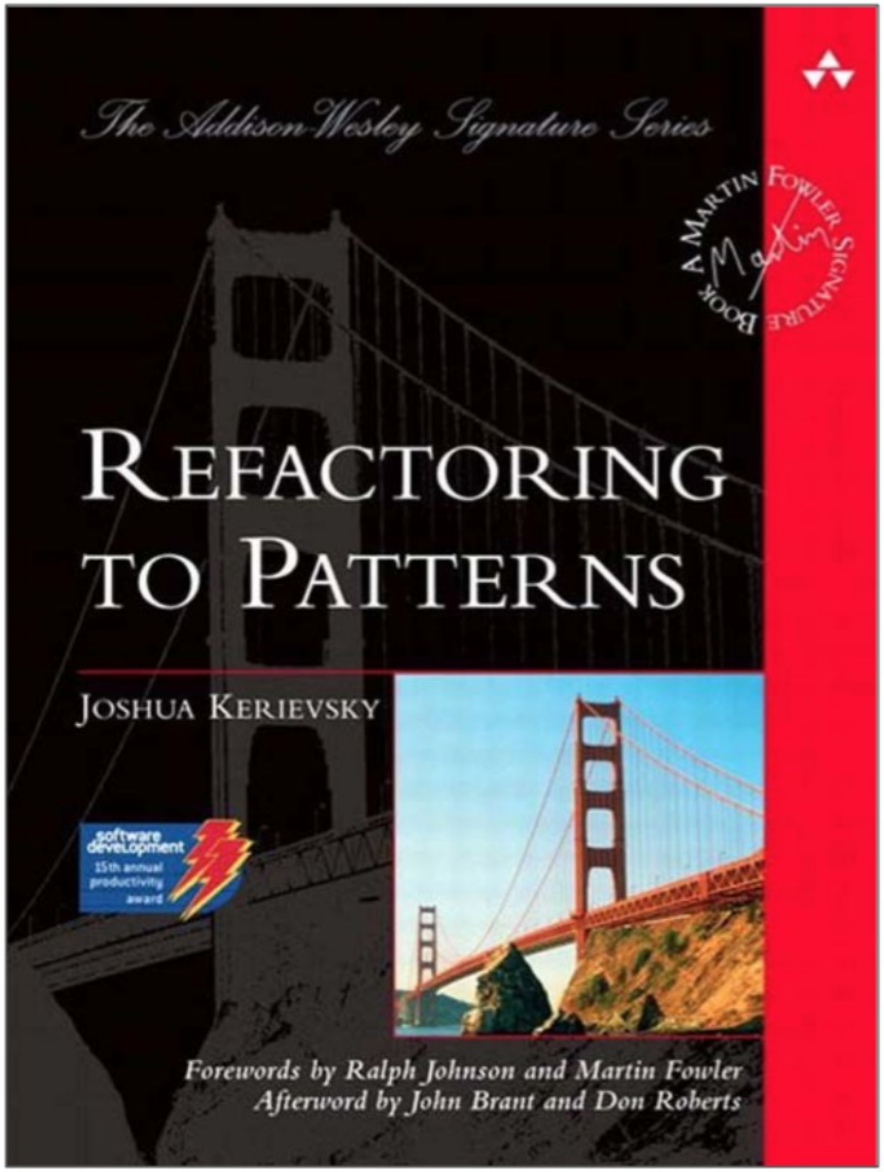
Exemplo de código

```
void imprimeDivida() {
imprimirCabecalho();
double divida = calcularDivida();
imprimirDetalhes(divida);
}

private void imprimirDetalhes(double divida) {
System.out.println("nome: " + nome);
System.out.println("divida total: " + divida);
}

private double calcularDivida() {
double divida = 0.0;
int i = 0;
ArrayList<Itens> e = pedidos.getPedidos();
while (i < e.size()) {
Itens cada = (Itens) e.get(i);
divida += cada.getValor();
}
return divida;
}

private void imprimirCabecalho() {
System.out.println("*****************************");
System.out.println("*** Dívidas do Cliente ****");
System.out.println("*****************************");
}
```

Bibliografia



Bibliografia



Conceito

- Manutenção de Software: Todas as atividades, antes e depois da entrega, necessárias para proporcionar suporte de baixo custo ao software.

SWEBOK

Conceito

- Sub áreas SWEBOK
  - Fundamentos de Manutenção de Software (Definições e terminologias; Categorias de manutenção)
  - Questões Chave de Manutenção de Software (Estimativas de custos para a manutenção Métricas para a manutenção de software)
  - Processo de Manutenção (Processos de manutenção; Atividade de manutenção)
  - Técnicas de Manutenção (Engenharia reversa; **Refatoração**)

Conceito


Recursos Gastos com a Criação x Manutenção
Pressman (2006)
20%
80%
Criação
Manutenção

Conceito

Evolução dos Custos de Manutenção		
Referência	**Data/Época**	**% para manutenção**
Pressman	1970	35% a 40%
Leinz e Swanson	1976	60%
Pigoski	1980 – 1984	55%
Pressman	1980	60%
Schach	1987	67%
Pigoski	1985	75%
Frazer	1990	80%
Pigoski	1990	90%

Refatoração

- Uma [pequena] modificação no sistema que não altera o seu comportamento funcional, mas que torna o código mais fácil de ser entendido e menos custoso de ser alterado:
  - simplicidade
  - flexibilidade
  - Clareza
- O desempenho pode piorar, otimizar não é o objetivo da refatoração

O espírito da Refatoração

(refatoração)

Antes

Depois

Exemplos de refatoração

- Alterar nomes de variáveis, métodos e objetos para melhorar legibilidade do código
  - String calc_imp_rnd(int a, Txs tx, int rnd)

    ->
  - String calculaImpostoDeRenda(int ano, Imposto taxas, int rendimento)
- Mudar parâmetros de métodos para que fiquem mais claros
- Melhorar o design da arquitetura
- Eliminar duplicação de código
- Flexibilizar métodos para novos usos

Refatoração sempre existiu...

- Melhorar o código sempre foi uma prática implícita
- Com sua popularização (e aceitação acadêmica) surge um vocabulário de refatorações comuns e um catálogo compartilhado
- Assim podemos utilizá-las mais sistematicamente.
- Podemos aprender novas técnicas, ensinar uns aos outros.

Origens

- Surgiu na comunidade de Smalltalk nos anos 80/90.
- Desenvolveu-se formalmente na Universidade de Illinois em Urbana-Champaign.
- Grupo do Prof. Ralph Johnson.
  - Tese de PhD de William Opdyke (1992).
  - John Brant e Don Roberts:
    - ***The Refactoring Browser Tool***
- *Kent Beck (XP) na indústria.*

Catálogo de Refatorações

- [Fowler, 2000] “Refatoração”, contém 72 refatorações.
  - https://refactoring.com/catalog/
- Análogo aos padrões de projeto orientado a objetos [Gamma et al. 1995] (GoF)
- Vale a pena gastar algumas horas com este catálogo. (GoF é obrigatório, não tem opção).
- [Kerievsky, 2004] “Refactoring to patterns”, catálogo recente com 27 refatorações que aplicam padrões!
  - https://industriallogic.com/xp/refactoring/catalog.html

Por que refatorar?

- Poxa, apesar de meu código está “feio”, ele está funcionando…

Por que refatorar?

- Poxa, apesar de meu código está “feio”, ele está funcionando…
  - Verdade, o compilador não preocupa se o código está claro ou não

Por que refatorar?

- Poxa, apesar de meu código está “feio”, ele está funcionando…
  - Verdade, o compilador não preocupa se o código está claro ou não
- “Software bom é software que tem manutenção” - Haroldo (Amigo do professor)

Por que refatorar?

- Poxa, apesar de meu código está “feio”, ele está funcionando…
  - Verdade, o compilador não preocupa se o código está claro ou não
- “Software bom é software que tem manutenção” - Haroldo (Amigo do professor)
  - Ou seja, software envolve mudanças e é um humano quem o faz

Por que refatorar?

- Melhora o projeto do software
- Torna o software mais fácil de entender
  - Sem entender, pode introduzir erros ou bugs
- Ajuda a encontrar falhas
- Ajuda a programar mais rapidamente

Quando Refatorar

“When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature.”

- Refactoring: Improving the Design of Existing Code

Quando Refatorar

- Sempre há duas possibilidades:

  1) Melhorar o código existente.

  2) Jogar fora e começar do 0.

- É sua responsabilidade avaliar a situação e decidir quando é a hora de optar por um ou por outro.

Sintomas que o código merece ser refatorado

- Quando você encontra código antigo que não entende de primeira
- Ao ler código feito por outros programadores, e perceber que ele não está claro (durante revisões por exemplo)
- Quando precisa consertar uma falha, ou adicionar funcionalidade
- *Once and Only Once*

O Primeiro Passo em Qualquer Refatoração

- Antes de começar a refatoração, verifique se você tem um conjunto sólido de testes para verificar a funcionalidade do código a ser refatorado.
- Refatorações podem adicionar erros.
  - porém, como são feitas em pequenos passos, é fácil recuperar-se de uma falha
- Os testes vão ajudá-lo a detectar erros se eles forem criados.

Pequenos passos

- Em geral, um *passo de refatoração* é tão simples que parece que ele não vai ajudar muito.
  - Cada passo é trivial.
  - Demora alguns segundos ou alguns poucos minutos para ser realizado.
  - É uma operação sistemática e óbvia.
- Mas quando se juntam 50 passos, bem escolhidos, em seqüência, o código melhora radicalmente.

Formato de refatorações nos catálogos

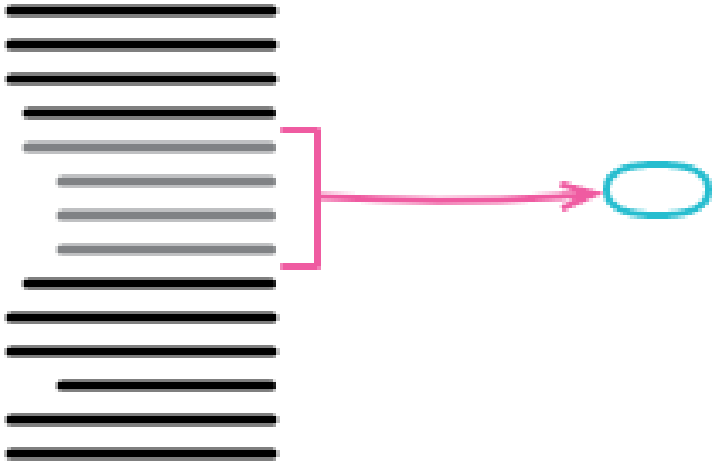
- **Nome** da refatoração.
- **Resumo** da situação na qual ela é necessária e o que ela faz.
- **Motivação** para usá-la (e quando não usá-la).
- **Mecânica**, i.e., descrição passo a passo.
- **Exemplos** para ilustrar o uso.

Obtendo Exemplos

- Pegar exemplo no github (https://github.com/diegoguedes/lab-refactoring/tree/master/catalogo-fowler)
- Para os testes, usaremos o JUnit para teste unitário e EclEmma para cobertura de código

Extrair Método (Extract Method)

- **Resumo:**
  - Você tem um fragmento de código que poderia ser agrupado. Mude o fragmento para um novo método e escolha um nome que explique o que ele faz.



Extrair Método (Extract Method)

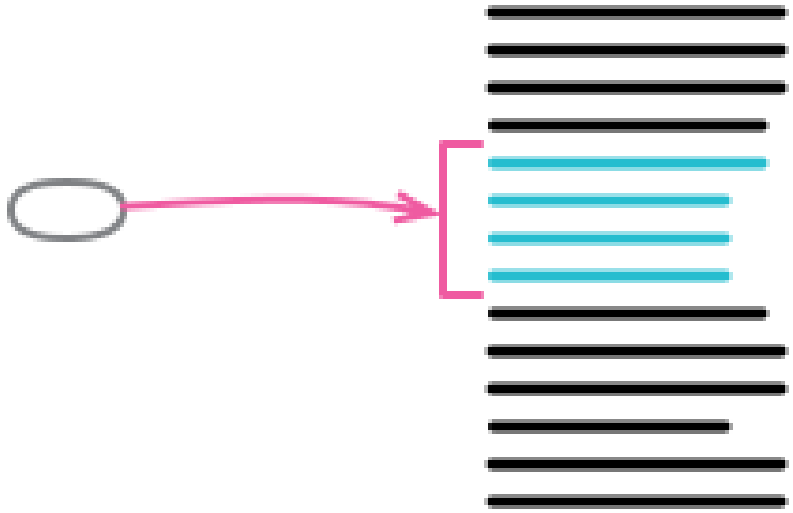
- **Motivação:**
  - Se um método é longo demais ou difícil de entender e exige muitos comentários, extraia trechos do método e crie novos métodos para eles. Isso vai melhorar as chances de reutilização do código e vai fazer com que os métodos que o chamam fiquem mais fáceis de entender. O código fica parecendo comentário.

Extrair Método (Extract Method)

- **Mecânica:**
  - Crie um novo método e escolha um nome que explicite a sua intenção (o nome deve dizer o que ele faz, não como ele faz).
  - Copie o código do método original para o novo.
  - Procure por variáveis locais e parâmetros utilizados pelo código extraído.
    - Se variáveis locais forem usados apenas pelo código extraído, passe-as para o novo método.
    - Caso contrário, veja se o seu valor é apenas atualizado pelo código. Neste caso substitua o código por uma atribuição.
    - Se é tanto lido quando atualizado, passe-a como parâmetro.
  - Compile e teste.

Internalizar Método (Inline Method)

- **Resumo:** a implementação de um método é tão clara quanto o nome do método. Substitua a chamada ao método pela sua implementação.



Internalizar Método (Inline Method)

- **Motivação:** bom para eliminar indireção desnecessária. Se você tem um grupo de métodos mau organizados, aplique *Internalizar Método* em todos eles seguido de uns bons *Extrair Métodos*.
- **Observação:** Note que é o inverso da “ *extract method*”

Internalizar Método (Inline Method)

- **Mecânica:**
  - Verifique se o método não é polimórfico ou se as suas subclasses o especializam
  - Ache todas as chamadas, e substitua pela implementação
  - Compile e teste
  - Remova a definição do método
  - Dica: se for difícil -> não faça.

Dividir variável temporária (split Temp variable)

- **Resumo:** Uma variável temporária atribuída mais que uma vez, mas que não está em um *loop* nem em uma coleção de variáveis temporárias

Dividir variável temporária (split Temp variable)

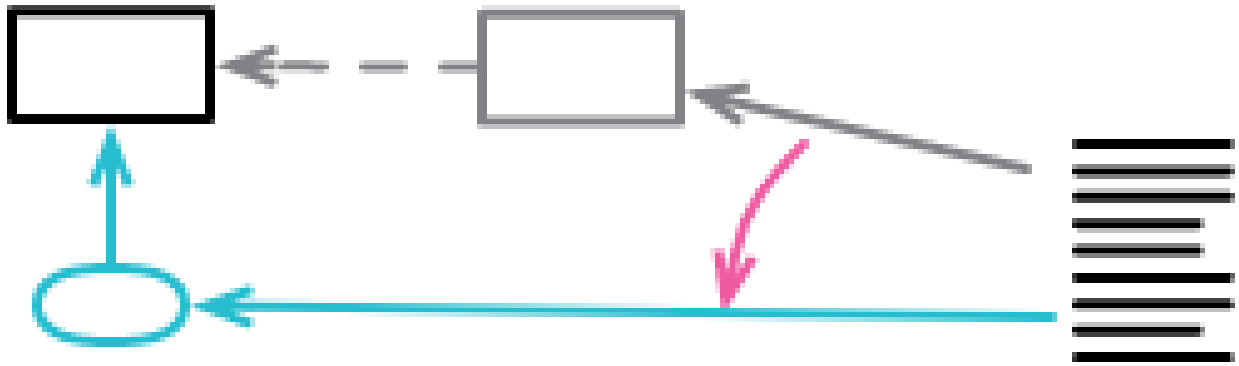
- **Motivação:** O código fica mais limpo e mais fácil de entender.

Dividir variável temporária (split Temp variable)

- **Mecânica:** Faça a separação de cada atribuição de variável temporária colocando o nome da real função da variável temporária.

Substituir variável temporária por consulta (Replace Temp with Query)

- **Resumo:** Uma variável local está sendo usada para guardar o resultado de uma expressão. Troque as referências a esta expressão por um método.



Substituir variável temporária por consulta (Replace Temp with Query)

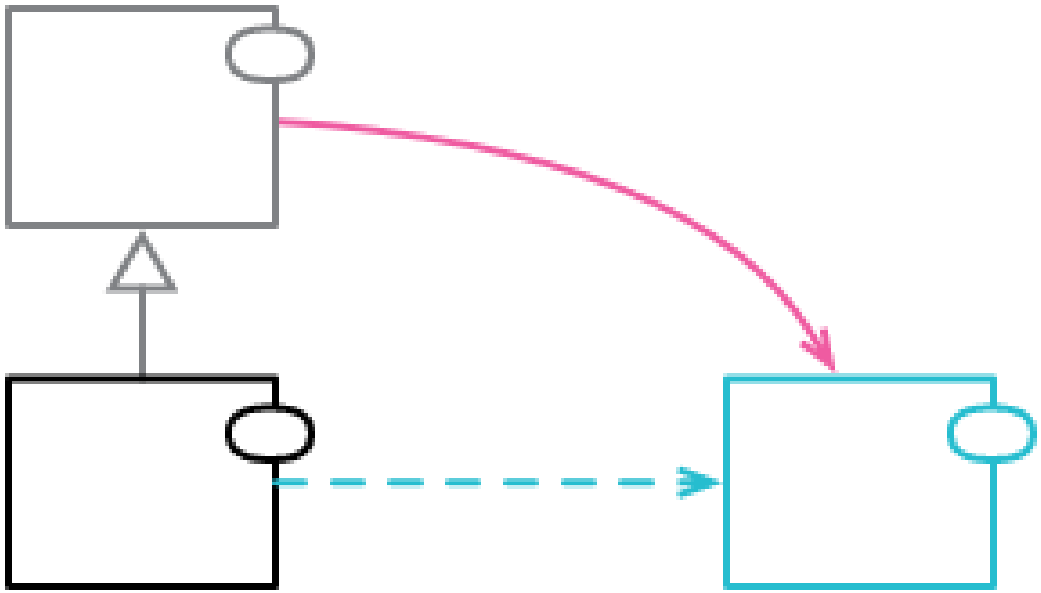
- **Motivação:** Variáveis temporárias encorajam métodos longos (devido ao escopo). O código fica mais limpo e o método pode ser usado em outros locais.

Substituir variável temporária por consulta (Replace Temp with Query)

- **Mecânica:**
  - Encontre variáveis locais que são atribuídas uma única vez
  - Se *temp* é atribuída mais do que uma vez use Dividir variável temporária (split Temp variable)
  - Declare *temp* como final Compile (para ter certeza)
  - Extraia a expressão
    - Método privado
  - Substitua a variável pelo método privado
  - Compile e teste

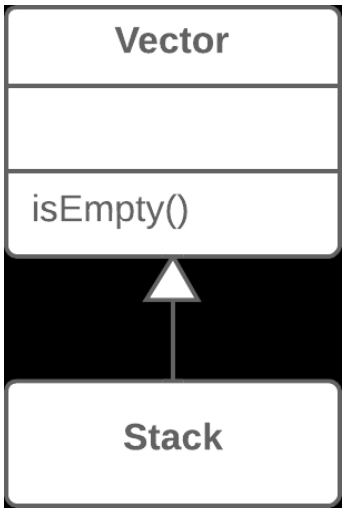
Substituir Herança por Delegação (*Replace Superclass with Delegation*)

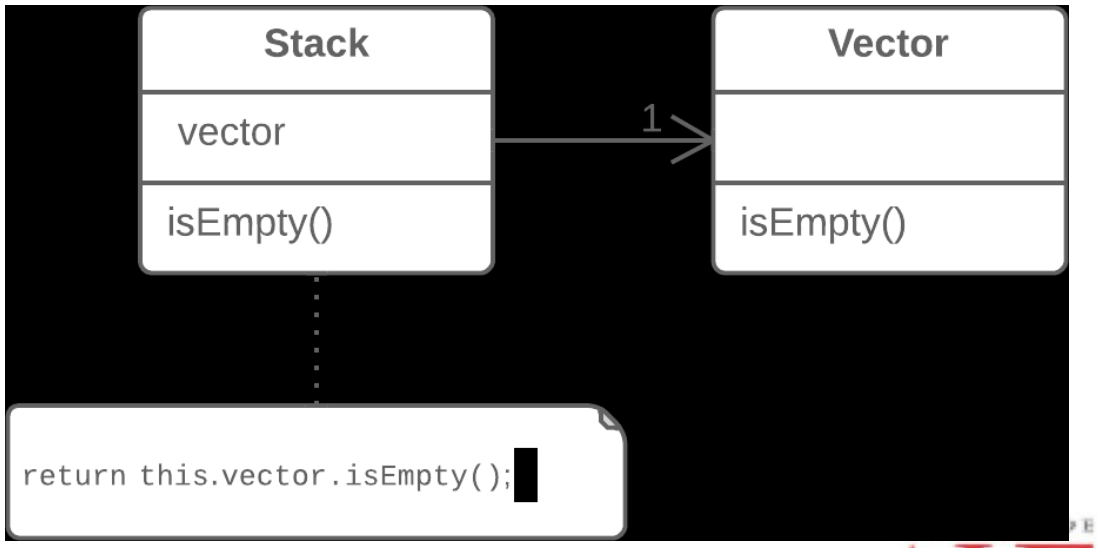
- **Resumo:** *Quando uma subclasse só usa parte da funcionalidade da superclasse ou não precisa herdar dados: na subclasse, crie um campo para a superclasse, ajuste os métodos apropriados para delegar para a ex-superclasse e remova a herança.*



Substituir Herança por Delegação (*Replace Superclass with Delegation*)

- **Motivação:** *herança é uma técnica excelente, mas muitas vezes, não é exatamente o que você quer. Às vezes, nós começamos herdando de uma outra classe mas daí descobrimos que precisamos herdar muito pouco da superclasse. Descobrimos que muitas das operações da superclasse não se aplicam à subclasse. Neste caso, delegação é mais apropriado.*



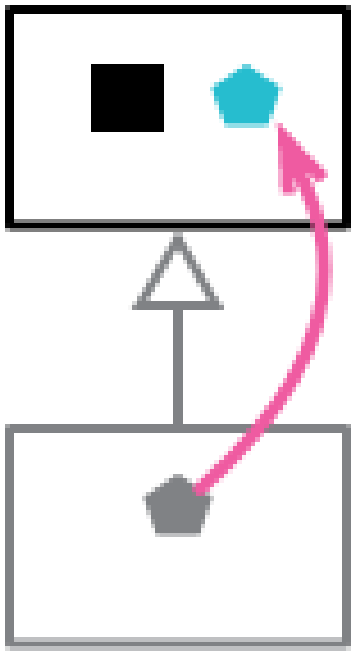


Substituir Herança por Delegação (*Replace Inheritance with Delegation*)

- **Mecânica**:
  - *Crie um campo na subclasse que se refere a uma instância da superclasse, inicialize-o com this*
  - *Mude cada método na subclasse para que use o campo delegado*
  - *Compile e teste após mudar cada método*
    - *Cuidado com as chamadas a super*
  - *Remova a herança e crie um novo objeto da superclasse*
  - *Para cada método da superclasse utilizado, adicione um método delegado*
  - *Compile e teste*

Condensar Hierarquia (Collapse Hierarchy)

- **Resumo:** *A superclasse e a subclasse não são muito diferentes. Combine-as em apenas uma classe.*



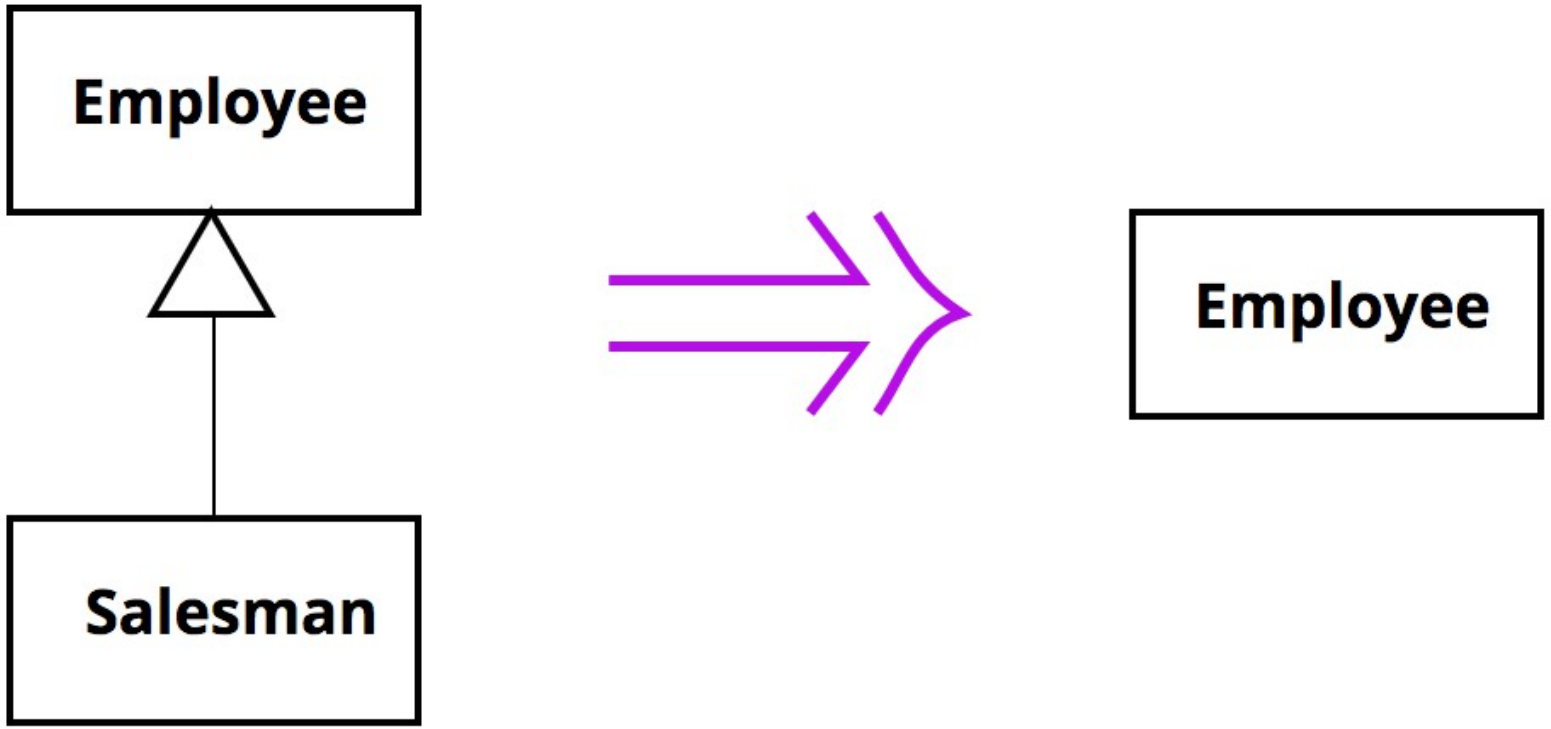
Condensar Hierarquia (Collapse Hierarchy)

- **Motivação:** *Depois de muito trabalhar com uma hierarquia de classes, ela pode se tornar muito complexa. Depois de refatorá-la movendo métodos e campos para cima e para baixo, você pode descobrir que uma subclasse não acrescenta nada ao seu desenho. Remova-a.*

Condensar Hierarquia (Collapse Hierarchy)

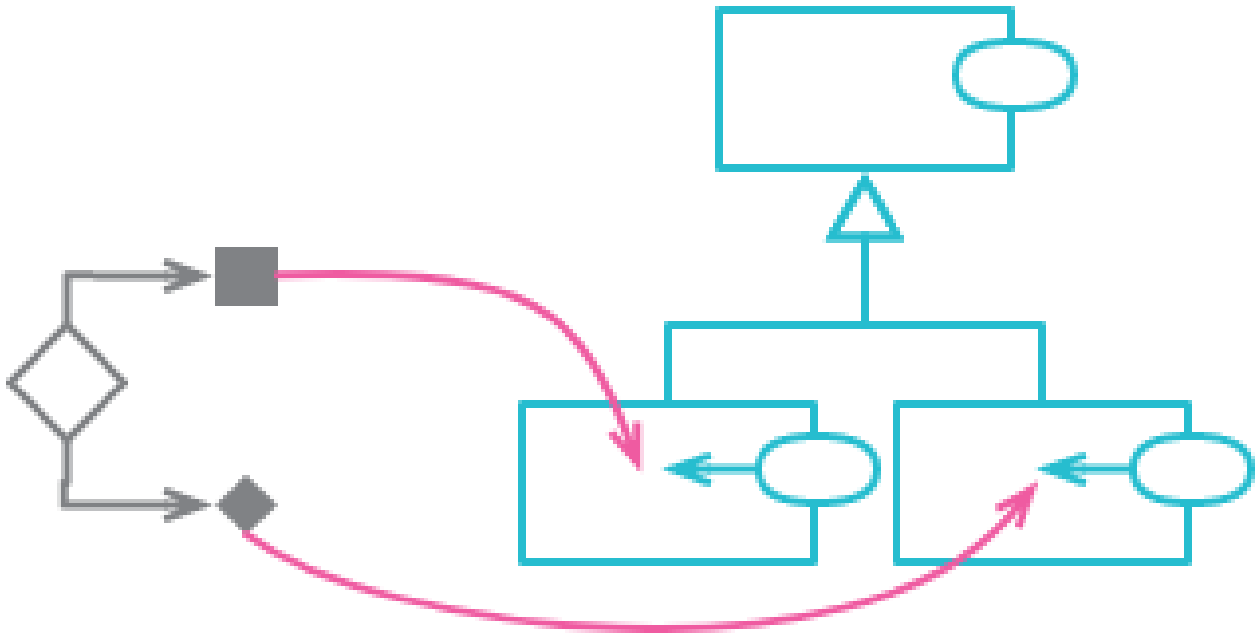
- **Mecânica:**
  - *Escolha que classe será eliminada: a superclasse ou a subclasse*
  - *Utilize uma refatoração para mover todo o comportamento e dados da classe a ser eliminada (e.g.: Mover Campo, Mover Método…)*
  - *Compile e teste a cada movimento*
  - *Ajuste as referências a classe que será eliminada*
  - *isto afeta: declarações, tipos de parâmetros e construtores.*
  - *Remove a classe vazia, compile e teste*

Condensar Hierarquia (Collapse Hierarchy)



Substituir comando condicional por polimorfismo (Replace Conditional with Polymorphism)

- **Resumo:** *Tem-se condições que escolhe diferentes comportamentos dependendo do tipo do objeto*



Substituir comando condicional por polimorfismo (Replace Conditional with Polymorphism)

- **Motivação:** O número de condições pode crescer significativamente e fica de difícil manutenção

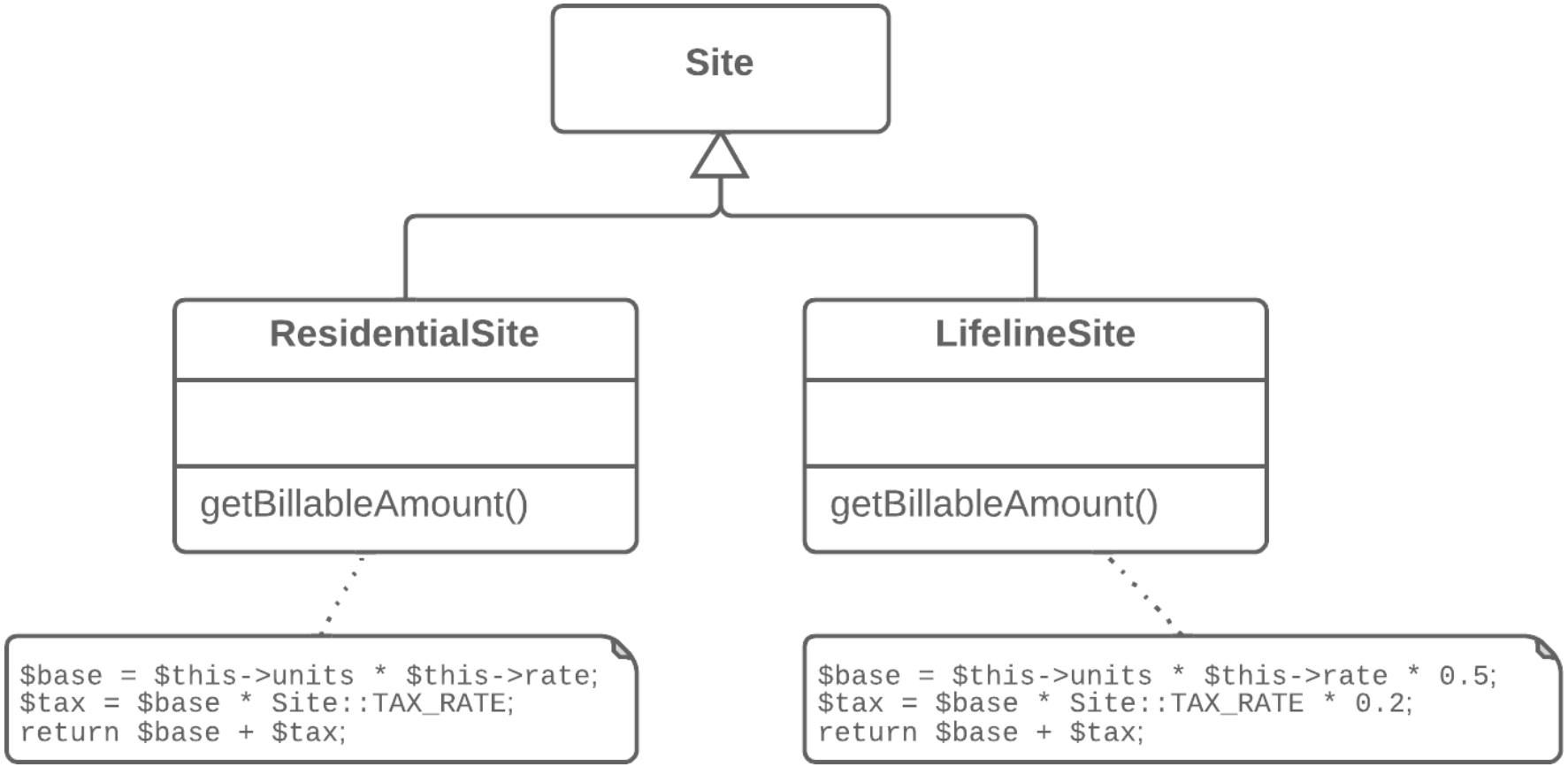
Substituir comando condicional por polimorfismo (Replace Conditional with Polymorphism)

- ***Mecânica:***
  - *Crie uma classe para cada condição*
  - *Estenda (crie uma herança) da classe original*
  - *Mova cada condicional para um método predominante na subclasse.*
  - *Torne o método original abstrato.*

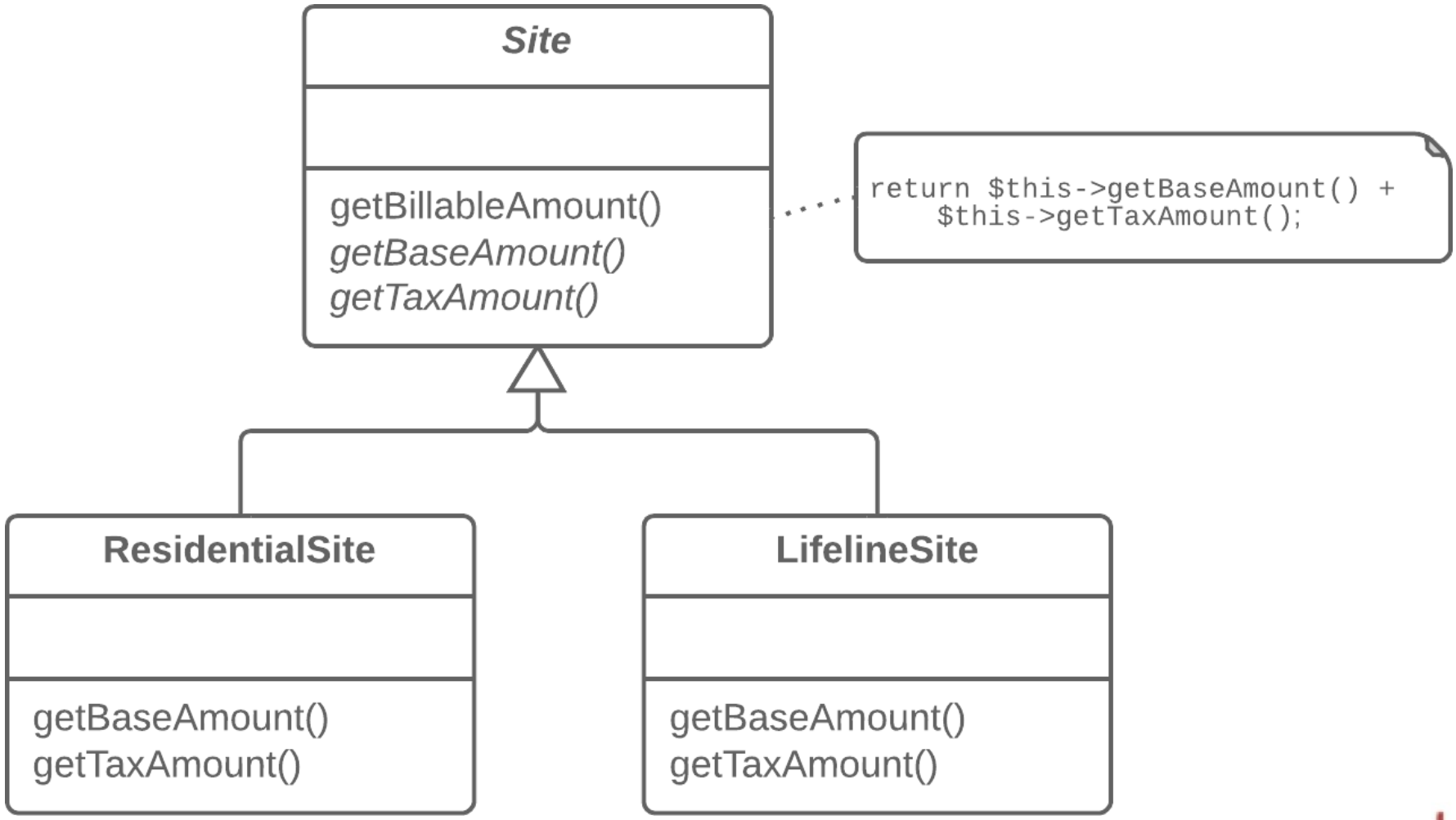
Método Molde (Form Template Method)

- **Resumo:** *Tem-se código duplicado*
- **Motivação:** Tem-se dois métodos em subclasses que executam etapas semelhantes na mesma ordem, mas as etapas são diferentes.

Método Molde (Form Template Method)



Método Molde (Form Template Method)



Método Molde (Form Template Method)

- **Mecânica:**
  - *Obtenha os passos em métodos com a mesma assinatura, para que os métodos originais se tornam os mesmos.*
  - *Crie uma superclasse abstrata, pegue o método e leve-o*

Encadeando Construtores (Pull up ConstructorBody)

- **Resumo:** *Seus construtores duplicam código*
- **Motivação:** Tem-se vários construtores que duplicam código
- **Mecânica:**
  - *Encadear os construtores em sequência para evitar ao máximo código repetido*

Sintoma básico

- Quando o código cheira mal, refatore-o!

Cheiro	Refatoração a ser aplicada
Código duplicado	Extrair Método Substituir o Algoritmo
Método muito longo	Extrair Método Substituir variável temporária por consulta Introduzir Objeto Parâmetro
Classe muito grande	Extrair Classe Extrair Subclasse Extrar Interface Duplicar Dados Observados
Intimidade inapropriada	Mover Método Mover Campo Substituir Herança por Delegação

Sintoma básico

Comentários	Extrair Método Introduzir Asserção
Muitos parâmetros	Substituir Parâmetro por Método Preservar o Objeto Inteiro Introduzir Objeto Parâmetro

Mais princípios básicos

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

Referências

- https://refactoring.com/catalog/
- https://refactoring.guru/
- Martin Fowler. *Refactoring: improving the design of existing code*.
- Kerievsky. *Refactoring to patterns*
- Melhorando a Qualidade de Código Pré-Existente. Cursos de Verão 2007 – IME/USP