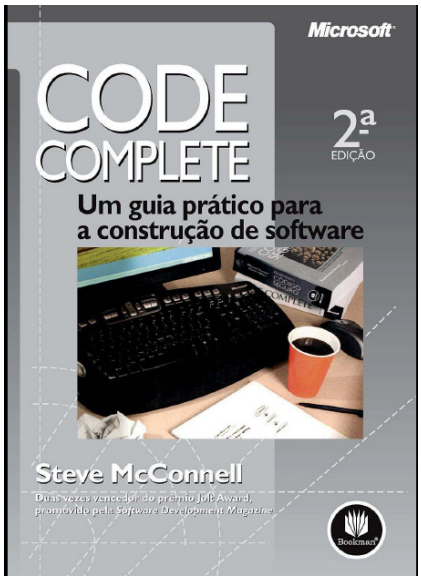
Construção de Software

Refatoração

Prof. Rubens de Castro Pereira, Me.
rubens@inf.ufg.br

INSTITUTO DE INFORMÁTICA
UFG



Code Complete
Um guia prático para a construção de software, 2ª ed., Steven McConnell
Bookman

Cap. 24 – Refatoração, pág. 589 a 610

Refatoração

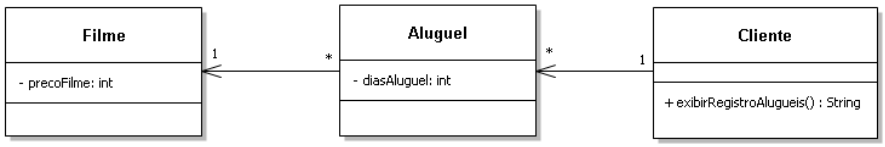
- "Uma alteração feita na estrutura interna do software para torná-lo mais fácil de ser entendido e mais barato de ser modificado, sem mudar seu comportamento visível" (Martin Fowler, 1999)
- "Máxima decomposição possível de um programa em partes constituintes" (Yourdon e Constantine, 1979)

Refatoração

- Considere uma pequena aplicação que visa controlar os aluguéis de uma vídeo-locadora, mais precisamente o código responsável por exibir o registro de aluguéis de um cliente. Algumas regras de negócio são:
  - Os filmes são divididos em faixas de preços: normal, infantil e lançamento;
  - O valor do aluguel depende da faixa de preço a que o filme pertence e do tempo que ele ficou alugado;
  - Cada aluguel rende um ponto para o cliente, sendo que se o filme for lançamento e o aluguel durar mais que dois dias ele ganha um ponto extra;
  - Com certa quantidade de pontos, o cliente ganha o aluguel de um filme.

Refatoração

Um diagrama de classes (parcial) da situação apresentada é:



Refatoração

- Faça a avaliação do código fonte das classes Filme.java, Aluguel.java e Cliente.java, visando refatorar as classes.

Motivos para Refatorar (*bad smells* – maus cheiros)

Motivo	Justificativa/Melhoria
1. Código duplicado	1. Copiar e colar é um erro
2. Rotina (método ou função) muito longa	2. Melhorar a modularidade
3. Loop muito longo ou aninhado com profundidade exagerada	3. Converter partes internas do loop em novas rotinas
4. Classe com fraca coesão	4. Subdividir a classe elevando a coesão
5. Interface de classe não fornece um nível de abstração consistente	5. Manter a integridade da interface
6. Lista de parâmetros excessivamente longa	6. Criar rotinas pequenas e bem definidas
7. Classe compartimentada	7. Dividir a classe conforme suas responsabilidades

Motivos para Refatorar (*bad smells* – maus cheiros)

Motivo	Justificativa/Melhoria
1. Alterações exigem modificações paralelas em várias classes	1. Reorganizar as classe de modo que alterações afetem uma classe
2. Instruções *case* precisam ser modificadas em paralelo	2. Talvez usar herança
3. Itens de dados relacionados e usados em conjunto não estão organizados em classes	3. Manipulação dos itens deve ser feita nas próprias classes
4. Uma rotina usa recursos de outra classe do que de sua própria classe	4. A rotina deve ser colocada na outra classe
5. Um tipo de dados é sobrecarregado	5. Crie classes simples
6. Uma classe pouco útil	6. Transfira as responsabilidades para outra classe e elimine-a
7. Um objeto intermediário sem muita utilidade	7. Elimine o objeto intermediário chamando diretamente as outras classes

Motivos para Refatorar (*bad smells* – maus cheiros)

Motivo	Justificativa/Melhoria
1. Uma rotina tem um nome insatisfatório	1. Altere o nome imediatamente
2. Os membros de dados são públicos	2. Oculte os membros e crie rotinas de acesso (*get/set*)
3. Uma subclasse usa apenas pequeno percentual das rotinas de suas classes ascendentes	3. Transforme a superclasse em dados-membros da subclasse inicial e crie as rotinas realmente necessárias
4. Uso de comentários para explicar um código difícil	4. Não documente um código ruim, mas reescreva-o
5. Há muitas varáveis globais	5. Tente isolar as variáveis globais em rotinas de acesso
6. Há código de configuração antes de uma chamada de uma rotina ou código de desmonte após uma chamada de rotina	6. Ajuste a lista de parâmetros de rotina
7. Um programa contém um código que pode ser necessário algum dia	7. Evite antecipar código

Motivos para NÃO Refatorar

- Refatoração não se refere a:
  - Correção de defeitos
  - Adição de novas funcionalidades
  - Modificação no projeto
- Refator produz alterações com propósitos bem definidos para a melhoria contínua

Refatorações Específicas

Nível de Dados

Nível de Instrução

Nível de Rotina

Implementações de Classe

Interface de Classe

NÍVEL DE SISTEMA

Refatorações em Nível de Dados

1. Substitua números mágicos por constantes nomeadas.
2. Torne o nome da variável mais claro ou mais informativo.
3. Mova uma expressão em linha, substituindo-a pela variável intermediária resultante dos cálculos.
4. Substitua uma expressão por uma rotina evitando duplicação de código.
5. Introduza uma variável intermediária para expressões.
6. Converta uma variável de uso múltiplo em diversas variáveis de uso único.
7. Para propósitos locais use uma variável local, em vez de parâmetro.
8. Converta uma primitiva de dados em uma classe, caso ela precise de comportamento adicional.
9. Converta um conjunto de códigos de tipo em uma classe ou enumeração ao invés de se ter constantes independentes.
10. Converta um conjunto de códigos de tipo em uma classe com subclasses.
11. Transforme um array em objeto.
12. Encapsule uma coleção.
13. Substitua um registro tradicional por uma classe de dados.

Refatorações em Nível de Instrução

1. Decomponha uma expressão booleana, simplificando-a e usando variáveis temporárias com bons nomes.
2. Mova uma expressão booleana complexa para uma função booleana com um bom nome.
3. Consolide trechos duplicados dentro de diferentes partes de uma condicional, movendo-as após o bloco *if-then-else.*
4. Use *break* ou *return* em vez de uma variável de controle de loop.
5. Retorne assim que houver reposta da rotina, em vez de atribuir um valor de retorno dentro das instruções *if-then-else* aninhadas.
6. Crie e use objetos nulos, em vez de testar diretamente valores nulos nos atributos da classe.

Refatorações em Nível de Rotina (função, procedimento ou método)

- Extraia código em linha e crie novas rotinas.
- Converta uma rotina longa em uma classe seguida de refatorações.
- Substitua um algoritmo complexo por um simples, se possível.
- Adicione um parâmetro quando a rotina necessitar de mais informações.
- Remova um parâmetro quando a rotina não utilizá-lo.
- Separe as operações de consulta das operações de modificações, pois as consultas não podem alterar o estado de um objeto.

Refatorações em Nível de Rotina (função, procedimento ou método)

- Combine rotinas semelhantes parametrizando-as. Quando duas rotinas diferirem apenas a um valor constante utilizado, passe-o como parâmetro.
- Separe as rotinas cujo comportamento dependa dos parâmetros passados quando a rotina executa códigos diferentes dependendo do valor dos parâmetros.
- Passe um objeto inteiro, em vez de vários campos específicos do mesmo objeto.
- Passe campos específicos, em vez de um objeto inteiro, quando criar um objeto apenas para passar para uma rotina e usando poucos atributos do mesmo.

Refatorações em Nível de Rotina (função, procedimento ou método)

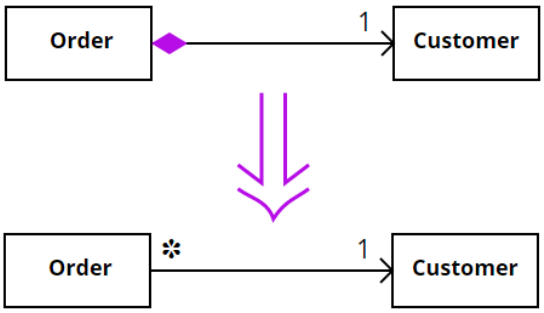
- Encapsule os *downcastings* quando uma rotina retorna um objeto, aplicável às rotinas que retornam iteradores, coleções e elementos de coleções:
  - Altere o tipo do objeto de retorno passando de genérico para o tipo específico.

```
Object ultimoItem() {
  return itens.lastElement();
}
```

```
Reading ultimoItem() {
  return (Reading) itens.lastElement();
}
```

Refatorações de Implementação no nível de Classe

- Transforme objetos de valor ou itens de dados em objetos de referência quando houver numerosas cópias de objetos grandes e complexos. Objetos de referência endereçam o objeto mestre.



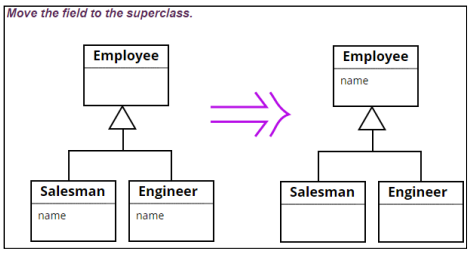
17

Refatorações de Implementação no nível de Classe

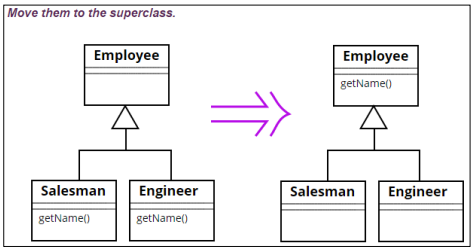
- Altere o posicionamento de rotina-membro ou dados-membro ao utilizar herança:
  - Puxar uma rotina para cima em sua superclasse
  - Puxar um campo para cima em sua superclasse
  - Puxar o corpo do construtor para cima em sua superclasse

Refatorações de Implementação no nível de Classe

Move the field to the superclass.



Move them to the superclass.



Create a superclass constructor; call this from the subclass methods.

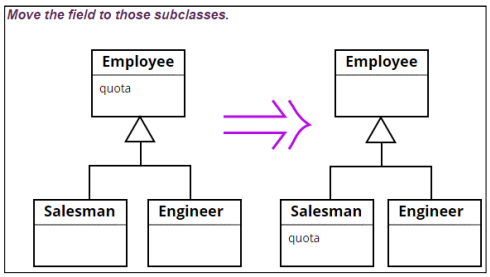
```
class Manager extends Employee...
  public Manager (String name, String id, int grade) {
    _name = name;
    _id = id;
    _grade = grade;
  }
```

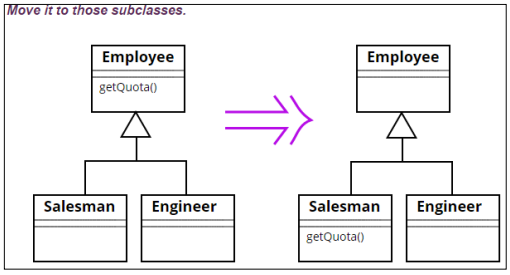
```
  public Manager (String name, String id, int grade) {
    super (name, id);
    _grade = grade;
  }
```


Refatorações de Implementação no nível de Classe

- Altere o posicionamento de rotina-membro ou dados-membro ao utilizar herança:
  - Puxar uma rotina para baixo em suas classes derivadas
  - Puxar um campo para baixo em suas classes derivadas
  - Puxar um construtor para baixo em suas classes derivadas

Refatorações de Implementação no nível de Classe

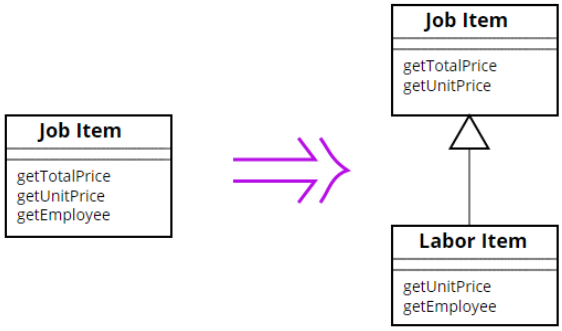




21

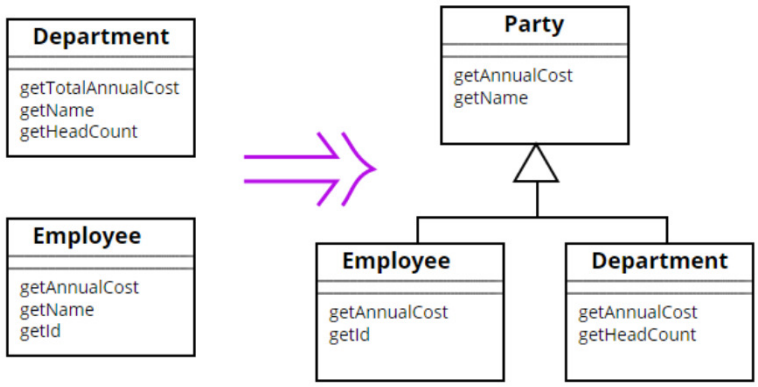
Refatorações de Implementação no nível de Classe

- Extraia um código especializado para uma subclasse quando o código é utilizado por um subconjunto de suas instâncias (objetos).



Refatorações de Implementação no nível de Classe

- Combine código semelhante em uma superclasse.



Refatorações em Nível de Sistema

- Crie uma fonte de referência definitiva para dados que não se pode controlar.
  - Exemplo de dados mantidos em um controle da interface gráfica da aplicação (GUI) que podem ser mantidos e controlados por uma classe utilizada por todas as demais.
- Transforme associação de classe unidirecional em associação de classe bidirecional quando as **duas classes precisam usar recursos uma da outra**.
- Transforme associação de classe bidirecional em associação de classe unidirecional quando **apenas uma classe precise usar recursos da outra classe**.
- Forneça um método de fábrica, em vez de construtor simples quando quiser trabalhar com objetos de referência em vez de objetos-valor.
- Substitua códigos de erro por exceções ou vice-versa conforme sua estratégia.

Refatorando com Segurança

- Use controle de versão para as configurações do projeto em desenvolvimento
- Mantenha as refatorações pequenas
- Faça uma refatoração por vez, recompilando e testando antes de iniciar a próxima refatoração
- Faça uma lista de passos que pretende executar
- Registre as novas refatorações a serem feitas durante um processo de refatoração
- Faça pontos de verificação frequentes, por meio de versionamentos
- Use os alertas do compilador e corrija-os
- Teste novamente após as revisões (refatorações)
- Adiciones casos de teste (unitários, ...) e remova os testes obsoletos
- Revise as alterações

Quando Refatorar é INOPORTUNO

- Não use refatoração como um disfarce para uma estratégia “codificar e corrigir”
- Refatorações referem-se a ***código que funciona***!
- Evite refatorar quando é mais prático reescrever

Estratégias de Refatoração

- Refatore quando adiciona uma **rotina**, verificando se elas estão bem organizadas entre si
- Refatore quando adicionar uma **classe**, verificando a existência de outras classes intimamente relacionadas com a nova classe
- Refatore quando **corrigir um defeito**, verificando a possibilidade de "melhorar" o código, com as devidas ressalvas
- Tenha como objetivo os **módulos propensos a erros** (merece maior atenção e, talvez, uma refatoração)
- Tenha com objetivo os **módulos de alta complexidade**
- Em um ambiente de manutenção, melhore as **partes que você alterar**

Refatoração

Leitura extra aula:

1. Code Complete – Um guia prático para a construção de software, 2ª ed., Steven McConnell, Bookman

   Parte V – Melhorias no código:
   - Cap. 24 – Refatoração, pág. 589 a 610

2. Refactoring: Improving the Design of Existing Code, 1999, Martin Fowler

3. Catálogo de refatorações: http://www.refactoring.com/catalog