

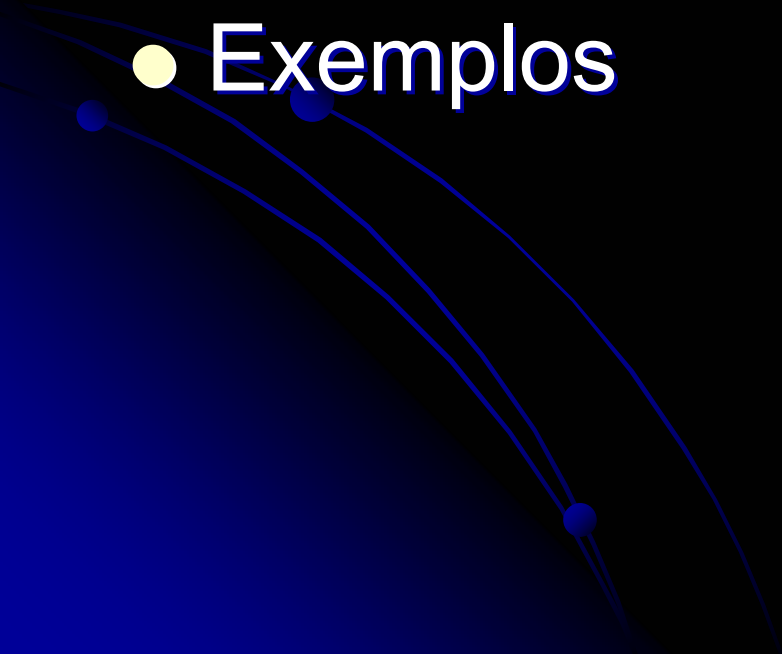
# Catálogo de Refatorações

Prof. Alberto Costa Neto

DComp/UFS

[alberto@ufs.br](mailto:alberto@ufs.br)

# Catálogo de Refatorações

- Nome
  - Resumo
  - Motivação
  - Mecânica
  - Exemplos
- 

# Classificação das Refatorações

- Adição de
  - Variáveis de instância, variáveis de classe, classe, método.
- Apagar
  - Variáveis de instância, variáveis de classe, classe, método
- Renomear
  - Variáveis de instância, variáveis de classe, classe, método\*, variáveis temporárias
- Renomear método
  - Renome simples, permuta argumentos, adicione argumentos, apague argumentos

# Classificação das Refatorações (2)

- Mover

- Push Up/Down

- Variável de Instância, Variável de Classe, Método.

- Mover para um Componente

- Método, Variável de Instância.

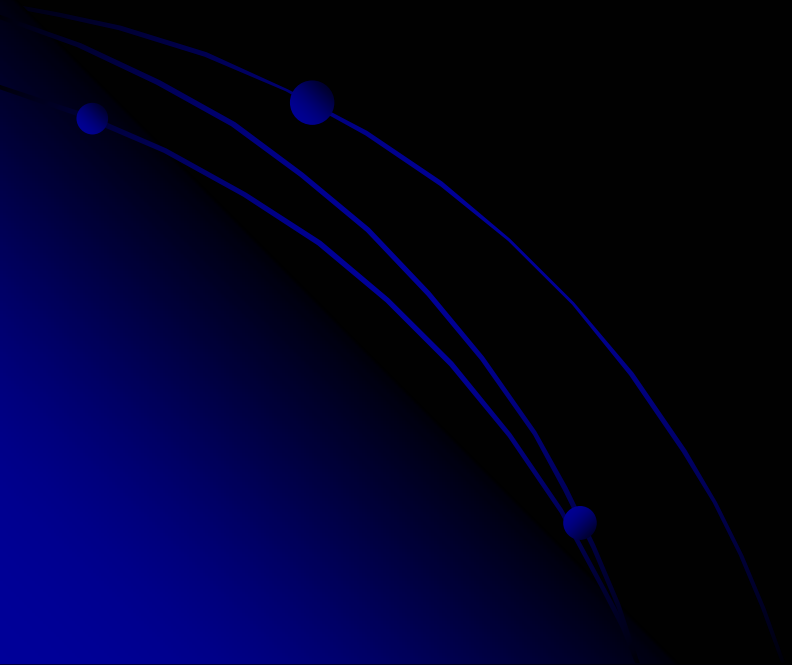
- Mude Superclasse

- Sub-métodos

- Extrair e Internalizar (*Inline*)

- Método, Variável Temporária

# Compondo Métodos



# Compondo Métodos

- Extrair Método, Internalizar (Inline) Método,
- Inline Temporário, Substituir Temporário por Consulta,
- Introduzir Variável Explicativa,
- Dividir Variável Temporária,
- Remover atribuição a parâmetros,
- Substituir Método por Objeto Método,
- Substituir Algoritmo

# Extrair Método

```
void imprimirDívida(double quantia) {  
    imprimirCabeçalho();  
    //imprime os detalhes  
    System.out.println("nome:" + _nome);  
    System.out.println("quantia:" + quantia);  
}
```



```
void imprimirDívida (double quantia) {  
    imprimirCabeçalho();  
    imprimirDetalhes(quantia);  
}  
void imprimirDetalhes (double quantia) {  
    System.out.println("nome:" + _nome);  
    System.out.println("quantia:" + quantia);  
}
```

# Mecânica para Extrair Método

- Crie um novo método e nomeie-o com a intenção do código extraído
- Copie o código extraído da fonte para o novo método
- Busque no código extraído referências a variáveis locais do código de origem.
- Se há variáveis locais usadas apenas dentro do código extraído, declare-as dentro do método novo
- Passe como parâmetro para o novo método as variáveis locais que sejam (somente) lidas pelo código extraído



# Mecânica para Extrair Método (2)

- Se alguma variável local for modificada
  - Verifique se pode extrair o código extraído como consulta
  - Se isto for difícil, não poderá aplicar Extrair Método. Use Dividir Variável Temporária ou Substituir Variável por Consulta e tente de novo
- Substitua o código extraído na origem por uma chamada ao novo método
- Compile e Teste

# Internalizar (*Inline*) Método

```
int getRating() {  
    return (moreThanFiveLateDeliveries()) ? 2 : 1;  
}  
boolean moreThanFiveLateDeliveries() {  
    return _numberOfLatgeDeliveries > 5;  
}
```



```
int getRating() {  
    return (_numberOfLatgeDeliveries > 5) ? 2 : 1;  
}
```

# Mecânica para Internalizar Método

- Certifique-se de que o método não é polimórfico
  - Não internalize se as subclasses sobrecarregarem o método
- Procure todas as chamadas ao método
- Substitua cada chamada com o corpo do método
- Compile e teste
- Apague a definição do método
- Compile e teste.

# Internalizar Temporário

```
double basePrice = anOrder.basePrice();  
return (basePrice > 1000)
```



```
return (anOrder.basePrice() > 1000)
```

# Mecânica para Internalizar Temporário

- Declare a variável temporária como *final*
  - Isto assegura que ela é realmente atribuída uma única vez
- Encontre todas as referências à variável e substitua com o lado direito da atribuição
  - Compile e teste após cada mudança
- Apague a declaração e a atribuição
- Compile e teste

# Substituir Temporário por Consulta

```
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;}  
}
```

# Mecânica para Substituir Temporário por Consulta

- Procure por uma variável que receba uma atribuição apenas uma vez
  - Se mais de uma vez, considere *Dividir Temporária*
- Declare a variável como *final*
- Compile
- Extraia o lado direito da atribuição para um método
  - Garanta que o método é livre de efeitos colaterais, senão use *Separar Consulta do Modificador*
- Compile e teste

# Introduzir Variável Explicativa

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&  
      (browser.toUpperCase().indexOf("IE") > -1) &&  
      wasInitialized() && resize > 0 ) {  
    // do something  
}
```



```
final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;  
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;  
final boolean wasResized = resize > 0;  
  
if (isMacOs && isIEBrowser && wasInitialized() && wasResized) {  
    // do something  
}
```



# Mecânica para Introduzir Variável Explicativa

- Declare uma variável temporária do tipo *final* e atribua a ela uma parte de uma expressão complexa
- Substitua a parte da expressão pela variável
  - Se a parte da expressão se repete, substitua todas as repetições, uma por vez
- Compile e teste
- Repita para outras partes da expressão

# Dividir Variável Temporária

```
double temp = 2 * (_height * _width);  
System.out.println (temp)  
temp = _height * _width;  
System.out.println (temp)
```



```
final double perimeter = 2 * (_height * _width);  
System.out.println (perimeter)  
final double area = _height * _width;  
System.out.println (area)
```

# Mecânica para Dividir Variável Temporária

- Troque o nome da temporária na sua declaração e na sua primeira atribuição
- Declare a nova temporária como *final*
- Altere as referências à temporária até a sua segunda atribuição
- Declare a temporária na sua segunda atribuição
- Compile e teste
- Repita os passos anteriores

# Remover Atribuições a Parâmetros

```
int discount (int inputVal, int quantity, int yearToDate) {  
    if (inputVal > 50) inputVal -= 2;  
}
```



```
int discount (int inputVal, int quantity, int yearToDate) {  
    int result = inputVal;  
    if (inputVal > 50) result -= 2;  
}
```

# Mecânica para Remover Atribuições a Parâmetros

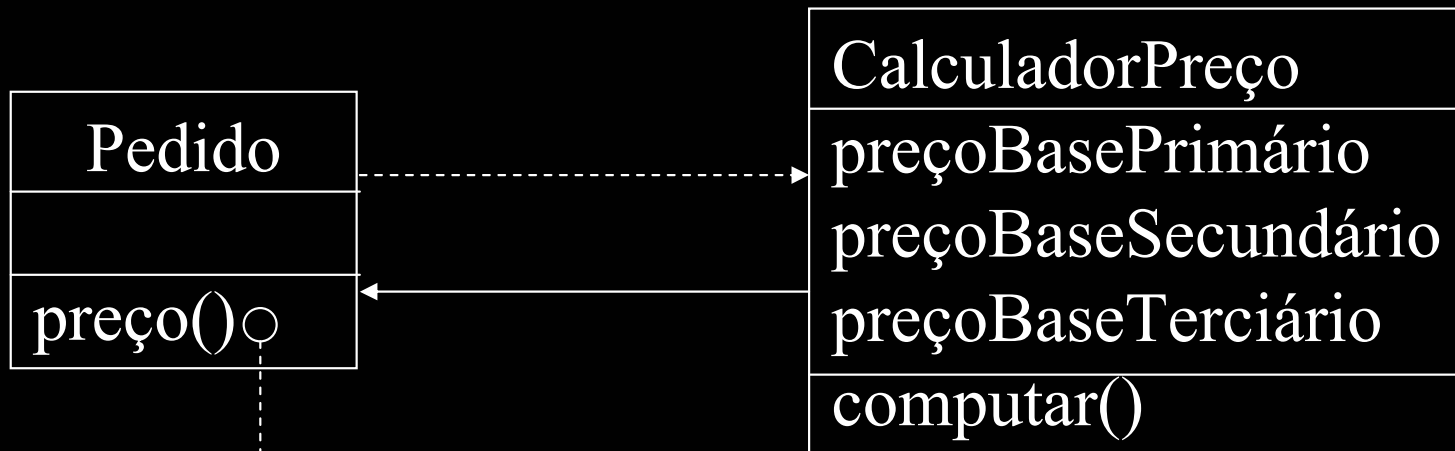
- Crie uma variável temporária para o parâmetro
- Substitua todas as referências ao parâmetro feitas após a atribuição à variável
- Substitua a atribuição para atribuir à nova variável
- Compile e teste

Mecânica para Java (passagem por valor)

# Substituir Método por Objeto Método

```
class Pedido ...  
    double preço() {  
        double preçoBasePrimário;  
        double preçoBaseSecundário;  
        double preçoBaseTerciário;  
        // computação longa  
        ...  
    }
```

# Substituir Método por Objeto Método(2)



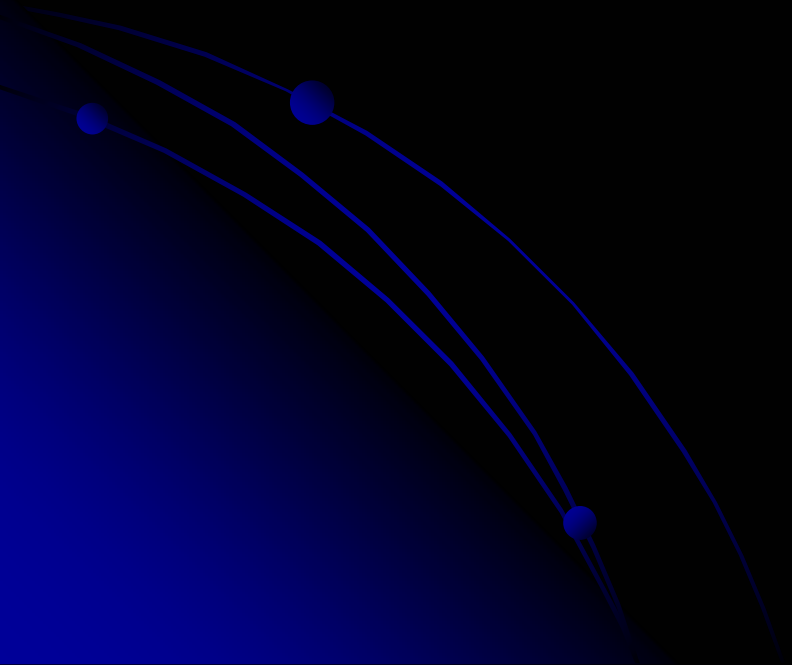
```
return new CalculadorPreço(this).computar()
```

# Mecânica para Substituir Método por Objeto Método

- Crie uma nova classe dando a ela o nome do método
- Adicione à nova classe um campo *final* para o objeto que hospedava o método original e um campo para cada v. temporária e cada parâmetro do método
- Adicione à nova classe um construtor que receba o objeto fonte e cada parâmetro
- Adicione à nova classe um método chamado “computar”
- Copie o corpo do método original para “computar”
- Compile
- Substitua o método velho pelo que cria o novo objeto e chama “computar”
- Compile e teste



# Movendo Recursos entre Objetos



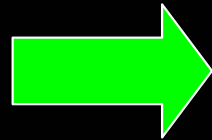
# Movendo Recursos entre Objetos

- Mover Método, Mover Campo,
- Extrair Classe, Internalizar Classe,
- Ocultar Delegação,
- Remover Intermediário,
- Introduzir Método Externo,
- Introduzir Extensão Local

# Mover Método

Classe 1
umMétodo()

Classe 1



Classe 2

Classe 2
umMétodo()

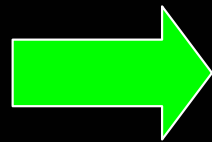
# Mecânica para Mover Método

- Examine todos os recursos usados pelo método fonte que estejam definidos na classe de origem. Considere se eles também devem ser movidos.
- Verifique nas sub e superclasses por outras definições do método
  - Se houver, talvez não possa mover a não ser que o polimorfismo seja expresso no destino
- Declare o método na classe de destino
  - Mude o nome se for preciso
- Copie o código da fonte para o destino e ajuste-o para funcionar
  - Se usa a classe de origem, passe a referência ao objeto de origem como um parâmetro novo
- Compile a classe de destino.

# Mecânica para Mover Método (2)

- Determine como referenciar o objeto destino correto a partir da classe de origem
  - Pode haver um campo, um parâmetro ou um método que lhe dará a referencia ao objeto destino
  - Senão, veja se pode criar facilmente um método que faça isto
  - Senão, precisa criar um novo campo na classe de origem que armazene esta referência
- Transforme o método fonte em um método de delegação
- Compile e teste
- Decida se irá remover o método de origem
  - Se decidir que sim, substitua todas as referências a esse método por referências ao método destino
- Compile e Teste

# Mover Campo

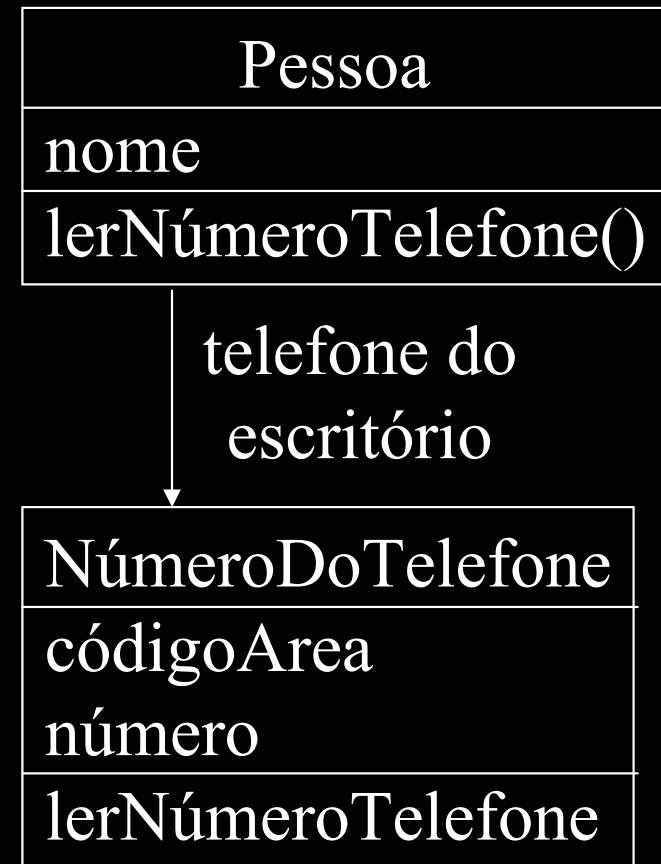
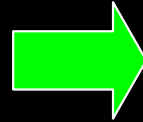


# Mecânica para Mover Campo

- Se o campo for público, use *Encapsular Campo*
- Crie um campo na classe destino com métodos get e set
- Compile a classe destino
- Determine como referenciar o objeto destino desde a fonte
- Apague o campo na fonte
- Substitua todas as referências ao campo na fonte com as referências ao método apropriado no destino

# Extrair Classe

Pessoa
nome
códigoÁreaEscritório
númeroEscritório
lerNúmeroTelefone()

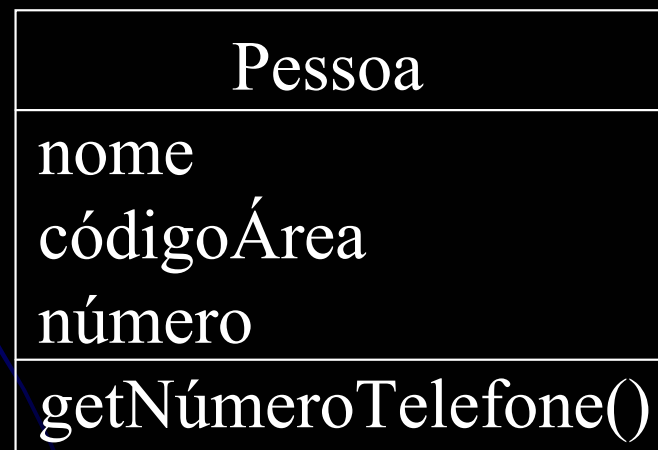
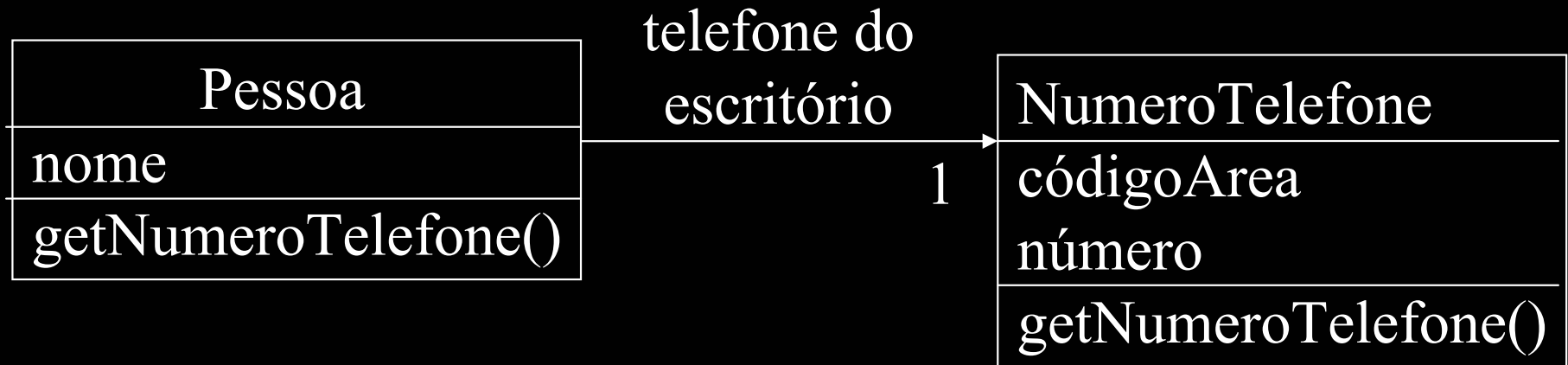




# Mecânica para Extrair Classe

- Decida como dividir as responsabilidades da classe
- Crie uma nova classe para dividir responsabilidades. Renomeie a classe antiga se for preciso
- Crie um relacionamento entre a classe nova e a antiga
- Use Mover Campo em cada campo que quiser mover
- Compile e Teste
- Use Mover Método em cada método desejado
- Compile e Teste
- Analise e reduza as interfaces de cada classe

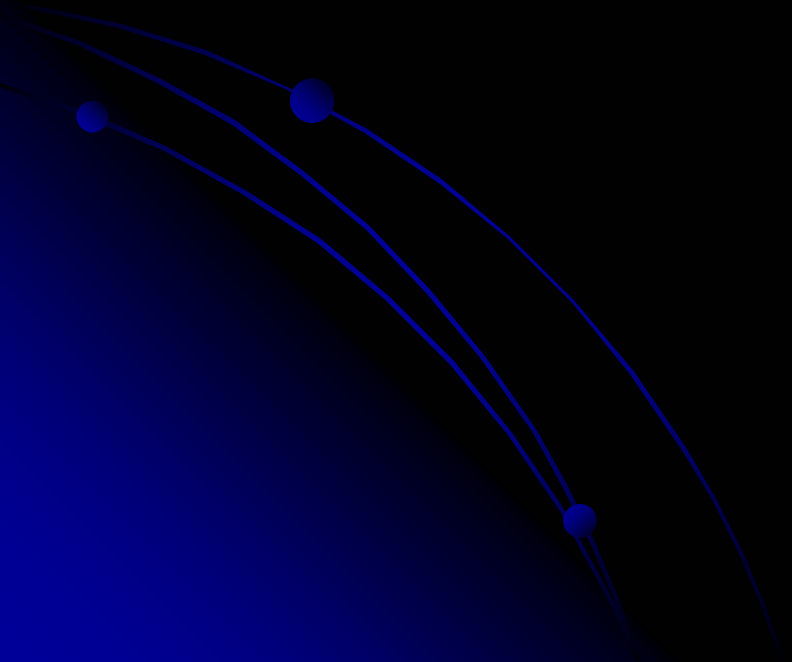
# Internalizar (Inline) Classe



# Mecânica para Internalizar Classe

- Declare o protocolo público da classe candidata na classe que vai absorvê-la
  - Delege todos os métodos desse protocolo à classe candidata
- Troque todas as referências à classe candidata para a classe que a absorverá
  - Troque o nome da classe candidata para que o compilador pegue as referências
- Compile e Teste
- Use Mover Método e Mover Campo da classe candidata para a que a absorve
- Apague a classe candidata

# Organizando Dados



# Organizando Dados

- Auto-Encapsular Campo, Substituir Atributo por Objeto, Mudar de Valor para Referência, Mudar de Referência para Valor, Substituir Vetor por Objeto, Duplicar Dados Observados, Transformar associação Unidirecional em Bidirecional, Transformar associação Bidirecional em Unidirecional, Substituir Número Mágicos, Encapsular Campo/Coleção, Substituir Registro por Classe de Dados, .....

# Auto-Encapsular Campo

```
private int _low, _high;  
boolean includes (int arg) {  
    return arg >= _low && arg <= _high; }
```

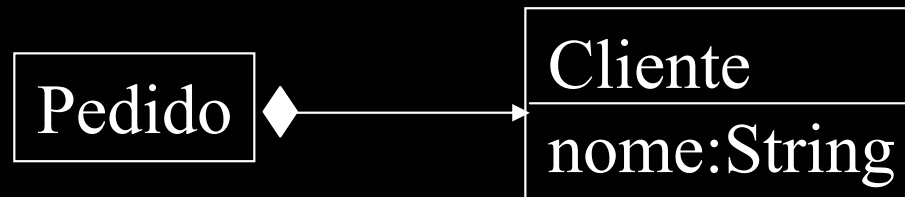


```
private int _low, _high;  
boolean includes (int arg) {  
    return arg >= getLow() && arg <= getHigh(); }  
int getLow() {return _low;};  
int getHigh() {return _high;};
```

# Mecânica

- Crie métodos get e set para o campo
- Encontre referências ao campo e substitua por chamadas get e set
- Torne o campo privado
- Compile e Teste

# Substituir Atributo por Objeto

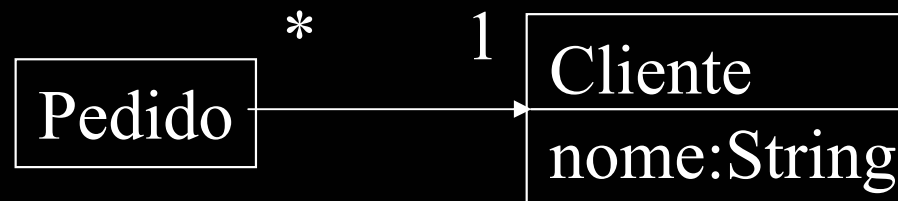
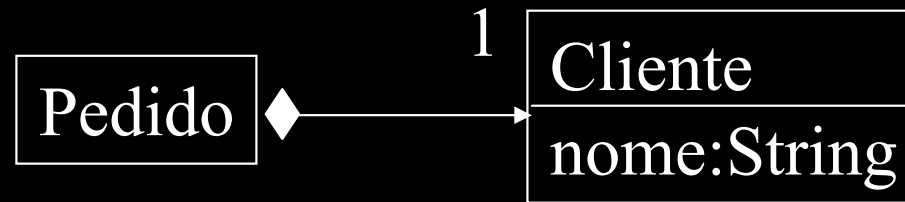




# Mecânica

- Cria a classe para o atributo e dentro dela crie um campo final para o atributo da classe original
- Compile
- Mude o tipo do campo na classe de origem para a nova classe
- Mude o método get na classe de origem para que chame o get da nova classe
- Se o campo é mencionado no construtor, atribua o campo usando o construtor da nova classe
- Troque o método de leitura para criar uma nova instância da nova classe
- Compile e Teste

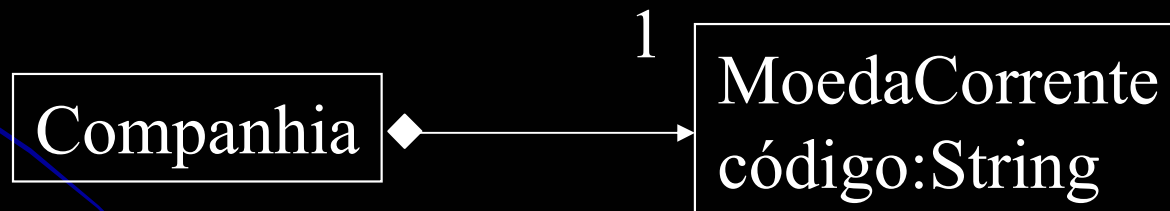
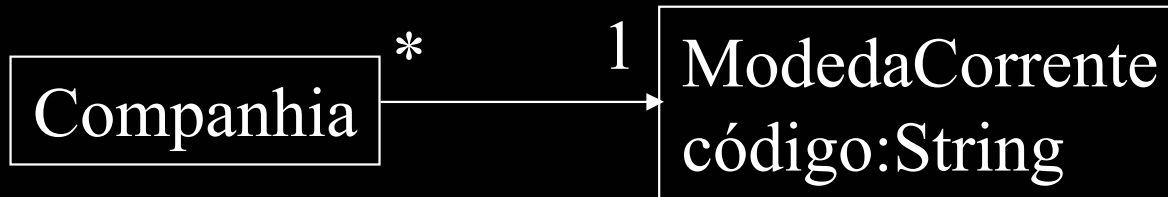
# Mudar Valor para Referência



# Mecânica

- Use Substituir o Construtor por um Método Fábrica
- Compile e Teste
- Decida que objeto é responsável pelo fornecimento de acesso aos objetos
- Determine se os objetos serão pré-criados ou criados dinamicamente
- Altere o método fábrica para retornar o objeto por referência
- Compile e Teste

# Mudar de Referência para Valor



# Mecânica

- Verifique se o candidato é imutável ou pode se tornar imutável
- Crie um método *equals* e um método *hash*
- Compile e Teste
- Considere remover qualquer método fábrica e criar um construtor público

# Substituir Vetor por Objeto

```
String[] row = new String[3];  
row[0] = "Liverpool";  
row[1] = "15";
```



```
Performance row = new Performance();  
row.setName("Liverpool");  
row.setWins("15");
```

# Mecânica

- Crie uma nova classe para representar a informação no vetor. Dê a ela um campo público para o vetor
- Altere todos os usuários do vetor para usarem a nova classe
- Compile e Teste
- Um a um, acrescente métodos set e get para cada elemento do vetor
- Crie um campo para cada elemento do vetor e mude os get's e set's para usarem o campo
- Compile e Teste
- Apague o vetor
- Compile e Teste

# Encapsular Campo

```
public String _name;
```



```
private String _name;  
public String getName() {return _name;};  
public void setName(String arg) {_name = arg;};
```



# Mecânica

- Crie métodos de gravação e leitura para o campo
- Encontre todos os clientes fora da classe que referenciam o campo. Substitua as referências por chamadas a get e set
- Compile e Teste
- Declare o campo privado
- Compile e Teste

# Encapsular Coleção

Pessoa
lerCursos():Set gravarCursos():Set



Pessoa
lerCursos():Unmodifiable Set adicionarCurso(:Curso) removerCurso(:Curso)

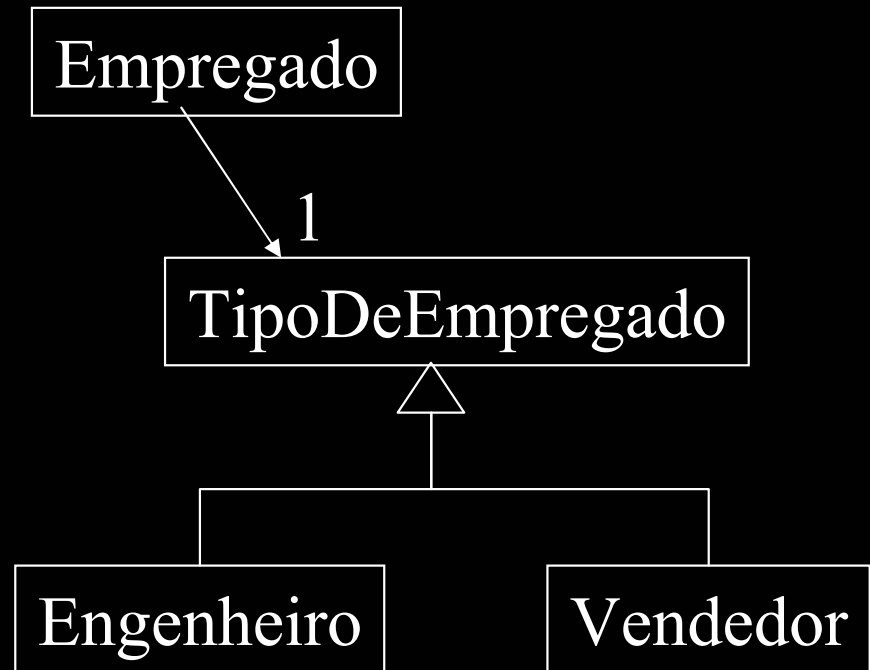
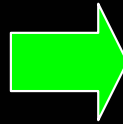
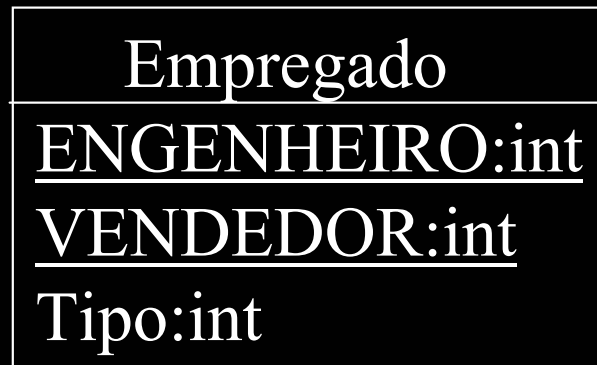
# Mecânica

- Acrescente métodos de adição e remoção para a coleção
- Inicialize o campo para uma coleção vazia
- Compile
- Descubra chamadas ao método de gravação. Modifique para usar operações de adição e remoção ou faça os clientes chamarem essas operações
- Compile e Teste
- Descubra todos os usuários do método de leitura que modifiquem a coleção. Altere-os para usarem os métodos de adição e remoção
- Compile e Teste

# Mecânica (2)

- Modifique o método de leitura para retornar uma visão apenas de leitura da coleção
- Compile e Teste
- Encontre os usuários do método de leitura. Procure por código que poderia estar no objeto que hospeda a coleção. Use Extrair Método e Mover Método para esse objeto

# Substituir Enumeração pelo Padrão State/Strategy



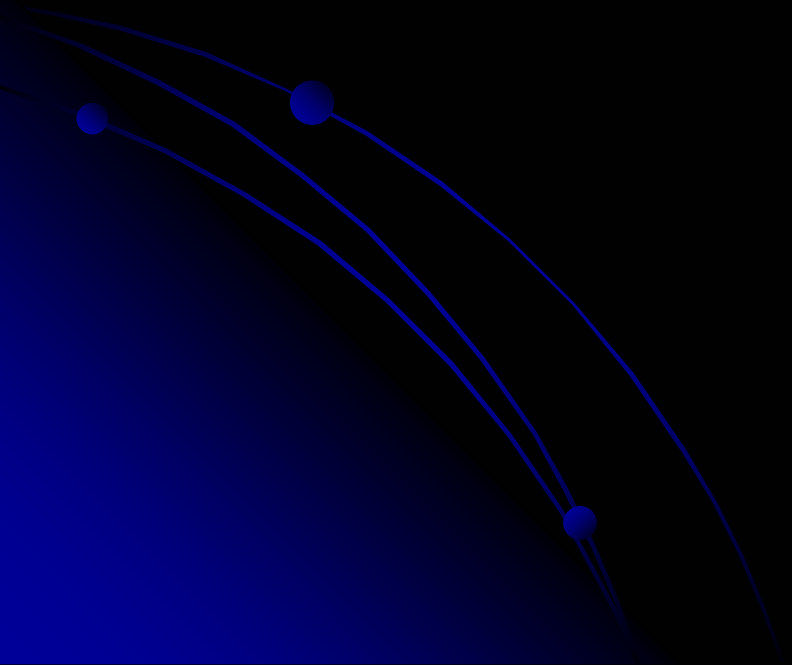
# Mecânica

- Auto-encapsule o campo enumerado
- Crie uma nova classe para o objeto estado. Dê um nome adequado
- Acrescente subclasses do objeto estado para cada valor possível do enumerado
- Crie uma pesquisa abstrata no objeto estado para retornar o valor da enumeração
- Compile

# Mecânica (2)

- Crie um campo na classe antiga para o novo objeto de estado
- Ajuste a pesquisa do valor do campo enumerado na classe original para delegar ao objeto estado
- Ajuste os métodos set do campo enumerado na classe original para atribuir uma instância da subclasse apropriada do objeto estado
- Compile e Teste

# Simplificando Condicionais





# Simplificando Condicionais

- Decompor Condicional, Consolidar Expressão Condicional, Consolidar Fragmentos Condicionais Duplicados, Remover *Flag* de Controle, Substituir Condição Aninhada por Cláusulas Guarda, Substituir Comando Condicional por Polimorfismo, Introduzir Objeto Nulo, Introduzir Asserção

# Decompor Condicional

```
if (date.before (SUMMER_START) || date.after (SUMMER_END))  
    charge = quantity * _winterRate + _winterServiceCharge;  
else charge = quantity * _summerRate;
```



```
if (notSummer(date))  
    charge = winterCharge(quantity)  
else charge = summerCharge(quantity);
```

# Mecânica

- Extraia o teste da condição para seu próprio método
- Extraia a parte do *then* a do *else* para seus próprios métodos

# Consolidar Expressão Condicional

```
double disabilityAmount() {  
    if (_seniority < 2) return 0;  
    if (_monthsDisabled > 12) return 0;  
    if (_isPartTime) return 0;  
    //compute the disability amount  
}
```



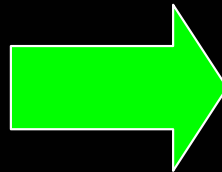
```
double disabilityAmount() {  
    if (isNotEligibleForDisability()) return 0;  
    //compute the disability amount  
}
```

# Mecânica

- Verifique se nenhuma das condições tem efeito colateral
- Substitua a cadeia de condicionais por uma única declaração condicional usando operadores lógicos
- Compile e Teste
- Considere usar Extrair Método sobre a condição

# Consolidar Fragmentos Condicionais Duplicados

```
if (isSpecialDeal()) {  
    total = price * 0.95;  
    send();  
}  
else {  
    total = price * 0.98;  
    send();  
}
```



```
if (isSpecialDeal())  
    total = price * 0.95;  
else  
    total = price * 0.98;  
    send();
```

# Mecânica

- Identifique o código comum, independente da condição
- Se estiver no início (fim), mova-o para antes (depois) da expressão condicional
- Se o código comum estiver no meio, verifique se o código anterior ou posterior a ele altera algo
- Se o código comum acontecer mais de uma vez, extraia um método

# Substituir Condição Aninhada por Cláusulas-Guarda

```
double getPayAmount() {  
    double result;  
    if (_isDead) result = deadAmount();  
    else {  
        if (_isSeparated) result = separatedAmount();  
        else {  
            if (_isRetired) result = retiredAmount();  
            else result = normalAmount; } }  
    return result;  
}
```





# Substituir Condição Aninhada por Cláusulas-Guarda (2)



```
double getPayAmount() {  
    if (_isDead) return deadAmount();  
    if (_isSeparated) return separatedAmount();  
    if (_isRetired) return retiredAmount();  
    return normalAmount; } }
```

# Mecânica

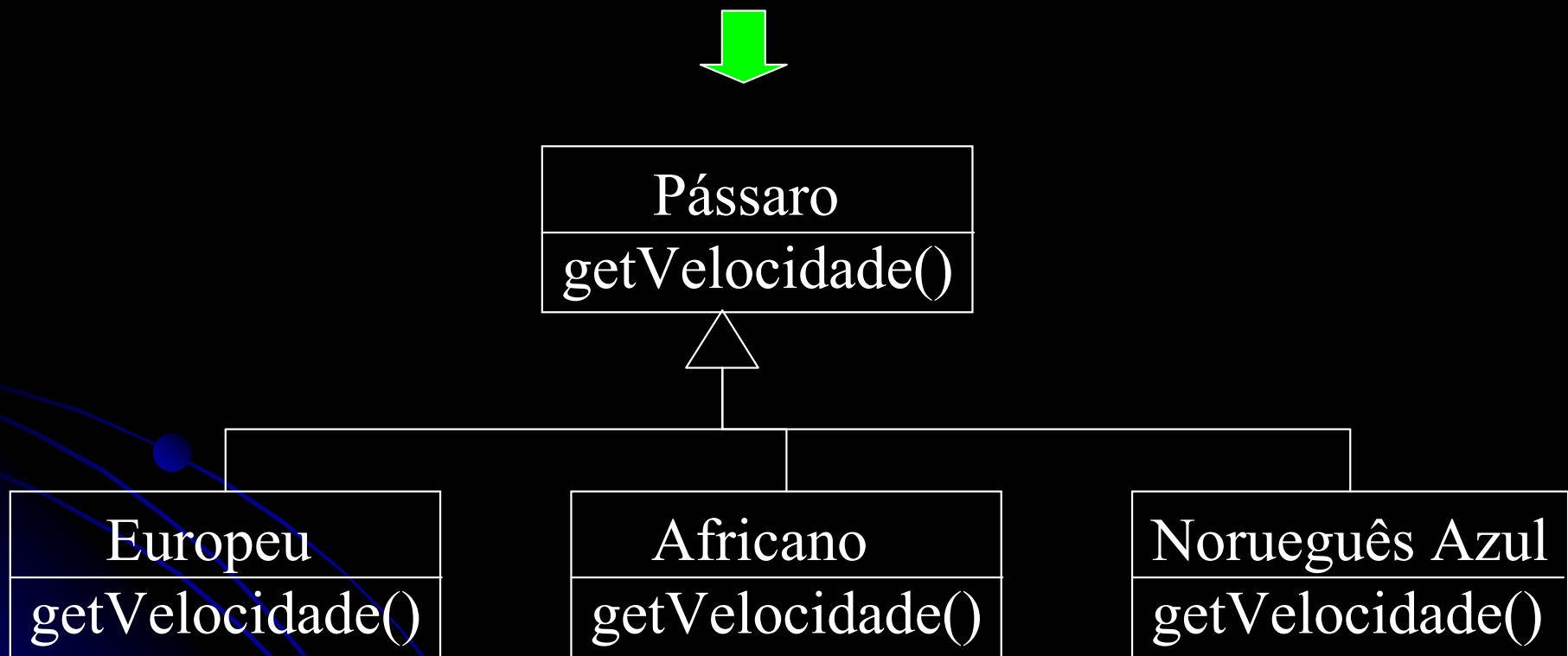
- Para cada verificação introduza a cláusula-guarda
- Compile e Teste após cada substituição
- Considere usar Consolidar Expressões Condicionais se as cláusulas-guarda produzirem o mesmo resultado

# Substituir Comando Condicional por Polimorfismo

```
double getVelocidade() {  
    switch (_tipo) {  
    case EUROPEU:  
        return getVeolcidadeBásica();  
    case AFRICANO:  
        return getVelocidadeBásica() – getFatorDeCarga() *  
            _numeroDeCocos;  
    case NORUEGUÊS_AZUL:  
        return: (_estaPregada) ? 0 : gerVelocidadeBásica(_voltage)  
    }  
}
```



# Substituir Comando Condicional por Polimorfismo (2)



# Substituir Comando Condicional por Polimorfismo

- Você tem um comando condicional que seleciona diferentes comportamentos de acordo com o tipo de um objeto
- *Mova cada ramificação do comando condicional para um método sobreposto em uma subclasse*
- *Torne abstrato o método original*

# Mecânica

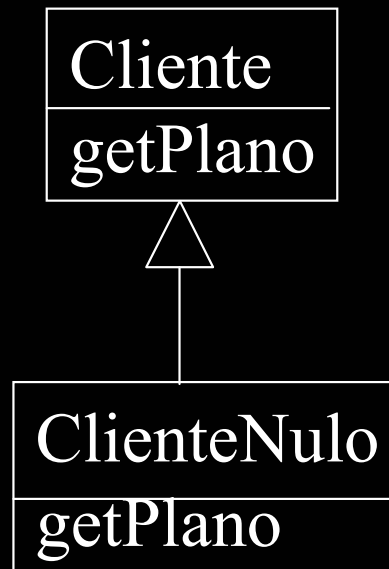
- Se não tiver a a estrutura de herança necessária, crie-a.
  - Use *Substituir Enumeração por Subclasse* ou *Substituir Enumeração pelo Padrão State/Strategy*
- Se a declaração condicional for parte de um método maior, separe-a com *Extrair Método*
- Se necessário, use *Mover Método* para colocar a expressão condicional no topo da hierarquia
- Selecione uma das subclasses. Sobrescreva o método da declaração condicional. Copie o corpo e ajuste
- Compile e Teste

# Mecânica (2)

- Remova a ramificação copiada da declaração condicional
- Compile e Teste
- Repita com cada ramificação
- Transforme o método da superclasse em abstrato

# Introduzir Objeto Nulo

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





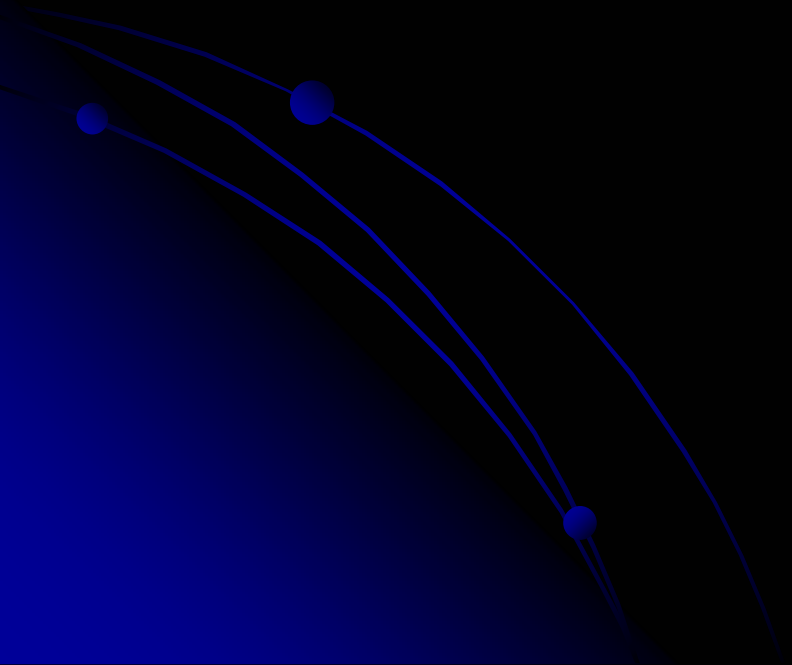
# Mecânica

- Crie uma subclasse da classe original para atuar com uma versão nula
  - Crie a operação `éNulo()` na classe original e na nula
- Compile
- Descubra todos os lugares que podem retornar um nulo. Substitua para retornarem um objeto nulo
- Encontre todos os lugares que comparam por *null* e substitua por uma chamada a `éNulo()`
- Compile e Teste

# Mecânica

- Procure código nos quais os clientes invocam uma operação no caso *not null* e outro comportamento no caso contrário
  - Sobrescreva a operação na classe nula com o comportamento
  - Remova o condicional com uma chamada polimórfica à operação
- Compile e Teste

# Tornando as Chamadas de Métodos Mais Simples

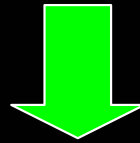


# Tornando as Chamadas de Métodos Mais Simples

- Renomear Método, Remover/Acrescentar Parâmetro, Separar a Pesquisa do Modificador, Parametrizar Método, Substituir Parâmetro por Métodos Explícitos, Preservar o Objeto Inteiro, Substituir Parâmetro por Método, Introduzir Objeto Parâmetro, Remover Método de Gravação (Setter), Ocultar Método, Substituir Construtor por um Método Fábrica Encapsular *Downcast*, Substituir Código de Erro por Exceção, Substituir Exceção por Teste

# Renomear Método

Cliente
getlimdcrat()



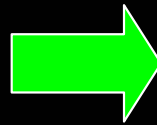
Cliente
getLimiteDeCréditoFaturável()

# Mecânica

- Verifique se o método é implementado por uma superclasse ou subclasse
  - Se for execute estes passos para cada implementação
- Declare um novo método com o nome novo. Copie o corpo e ajuste
- Compile
- Altere para que o antigo chame o novo
- Compile e Teste
- Encontre referências ao método antigo e atualize para o novo. Compile e Teste.
- Remova o método antigo
  - Pode querer manter o método antigo
- Compile e Teste

# Acrescentar Parâmetro

Cliente
getContato()



Cliente
getContato(:Date)

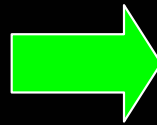
# Mecânica

- Similar ao de *Renomear Método*



# Remover Parâmetro

Cliente
getContato(:Date)



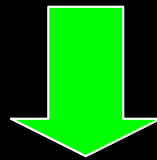
Cliente
getContato()

# Mecânica

- Similar a *Renomear Método*

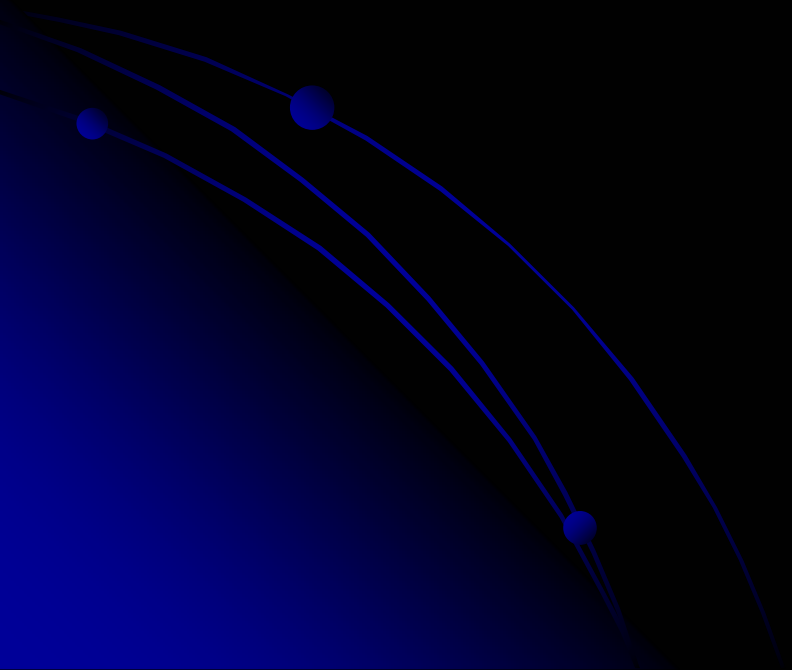
# Separar Consulta do Modificador

Cliente
getTotalPendenteEpreparaParaResumos()



Cliente
getTotalPendente() preparaParaResumos()

# Lidando Com Generalização

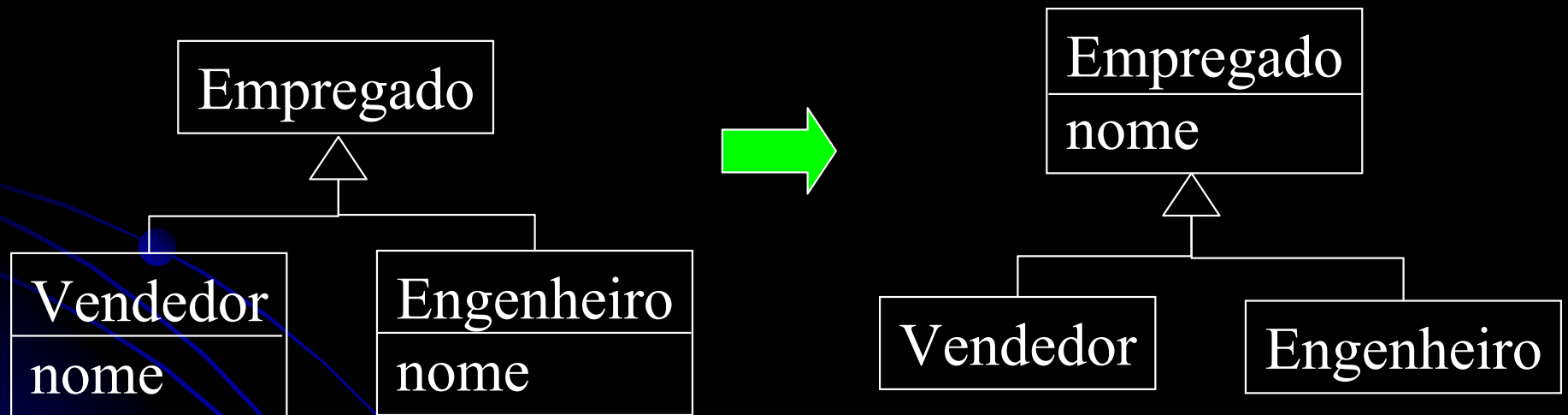


# Lidando Com Generalização

- Subir na Hierarquia
  - Campo, Método, Corpo de Construtor
- Descer na Hierarquia
  - Campo, Método
- Extrair
  - Subclasse, Superclasse, Interface
- Condensar Hierarquia
- Criar um Método Padrão (*Template Method*)
- Substituir Herança por Delegação
- Substituir Delegação por Herança

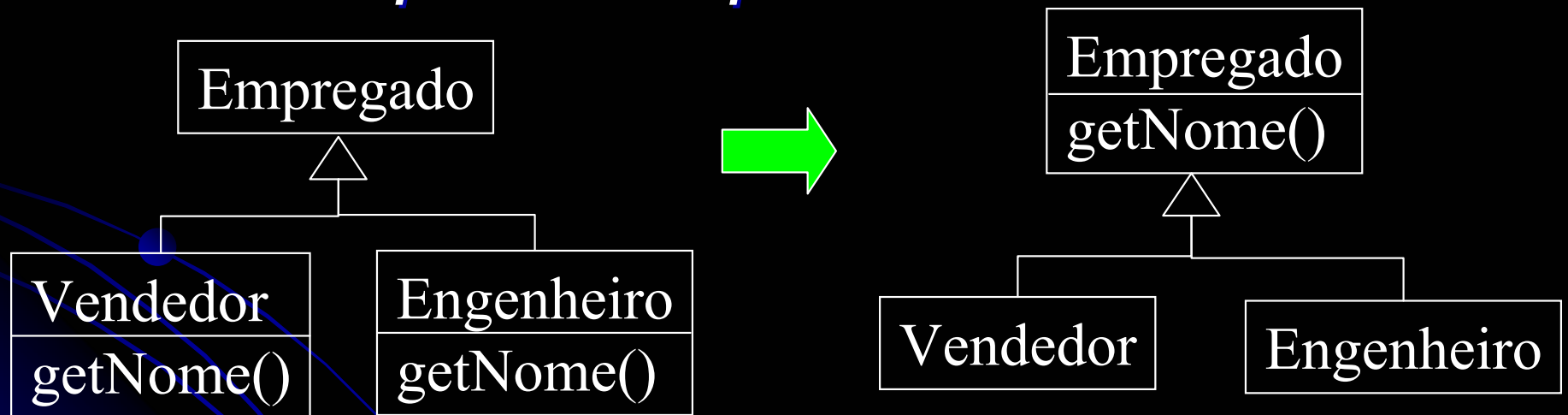
# Subir Campo na Hierarquia

- Duas Subclasse têm o mesmo campo
- *Mova o campo para a superclasse !*



# Subir Método na Hierarquia

- Duas subclasses tem métodos que produzem resultados idênticos
- *Mova-os para a superclasse*

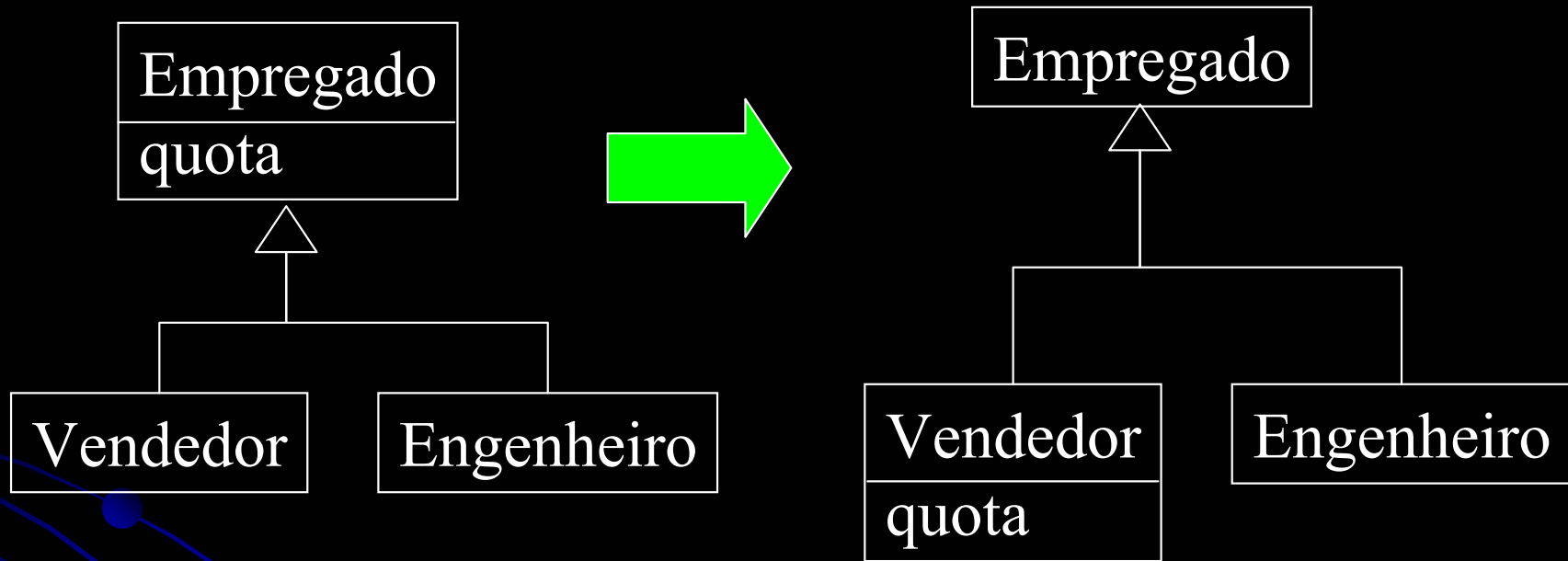


# Mecânica

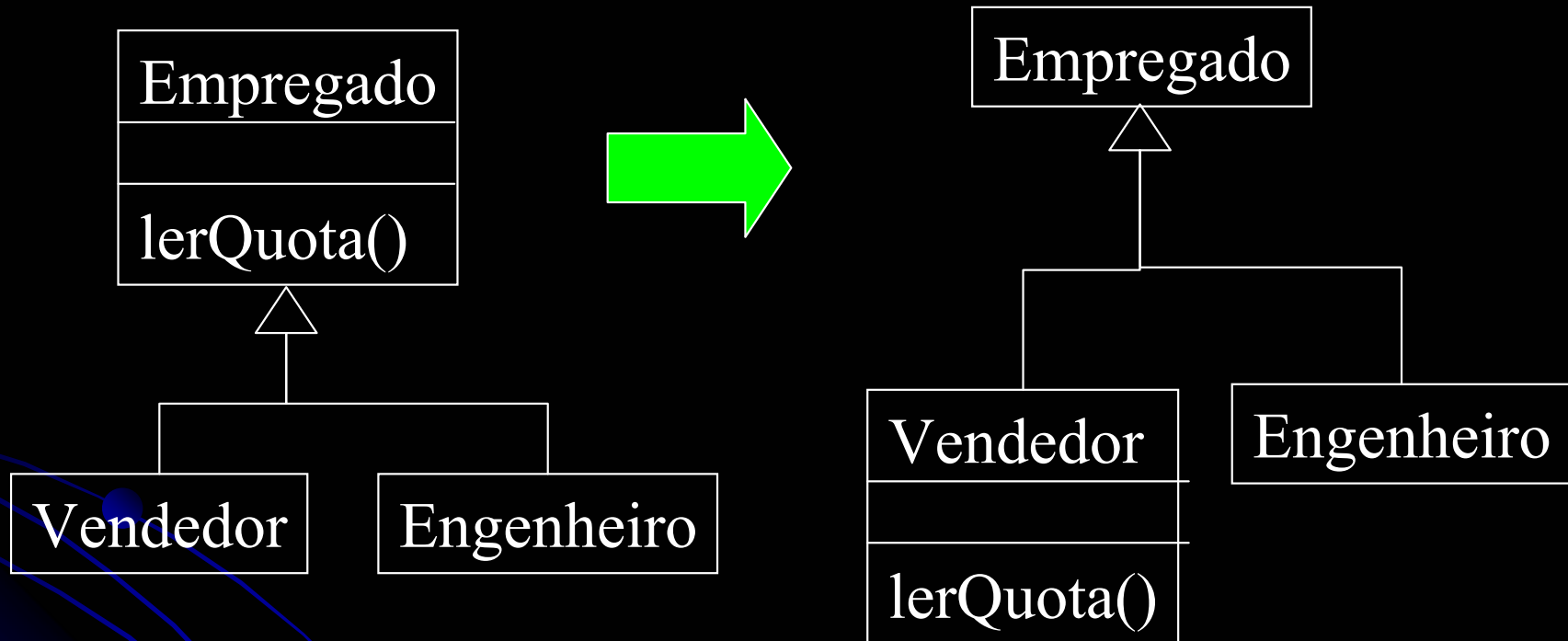
- Certifique-se de que os métodos sejam idênticos
  - Altere as assinaturas se for necessário
- Crie um novo método na superclasse, copie o corpo
- Apague na subclasse um por um, compile e teste



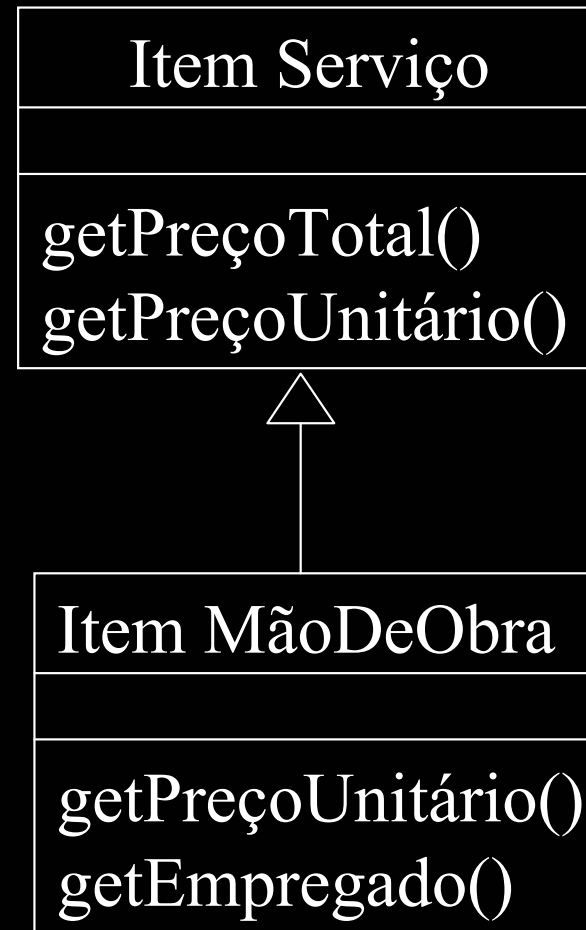
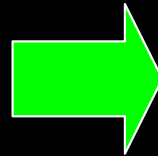
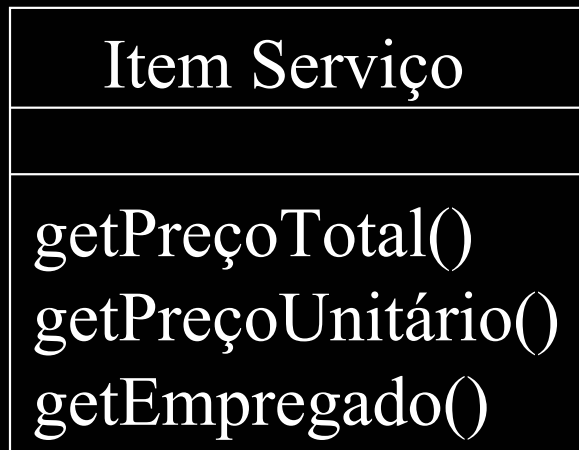
# Descer Campo na Hierarquia



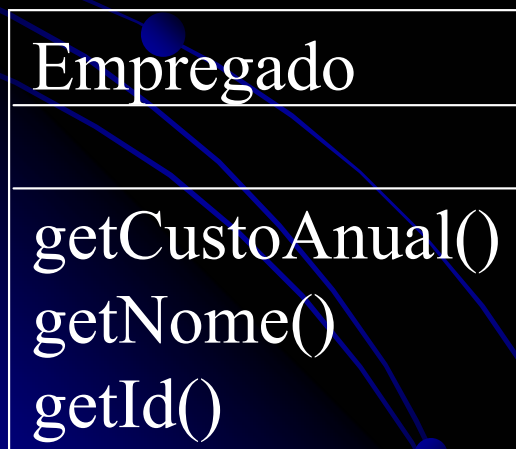
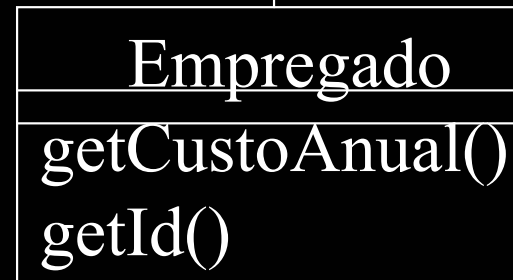
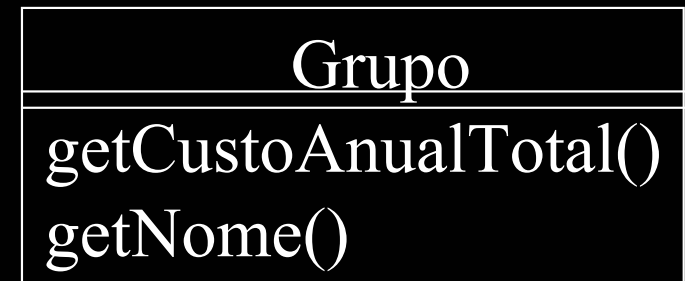
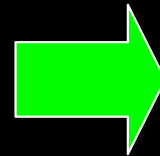
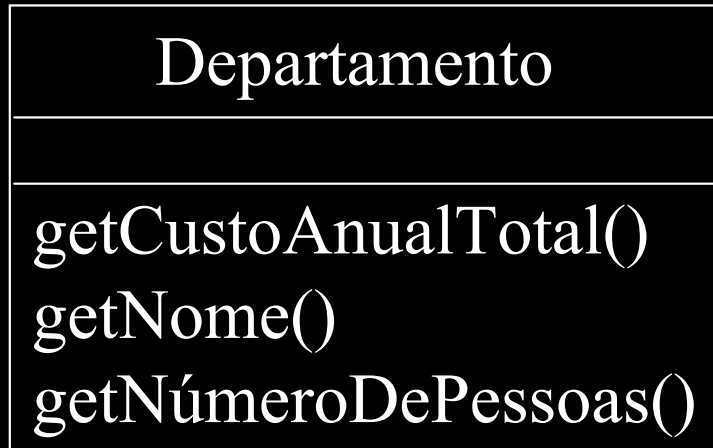
# Descer Método na Hierarquia



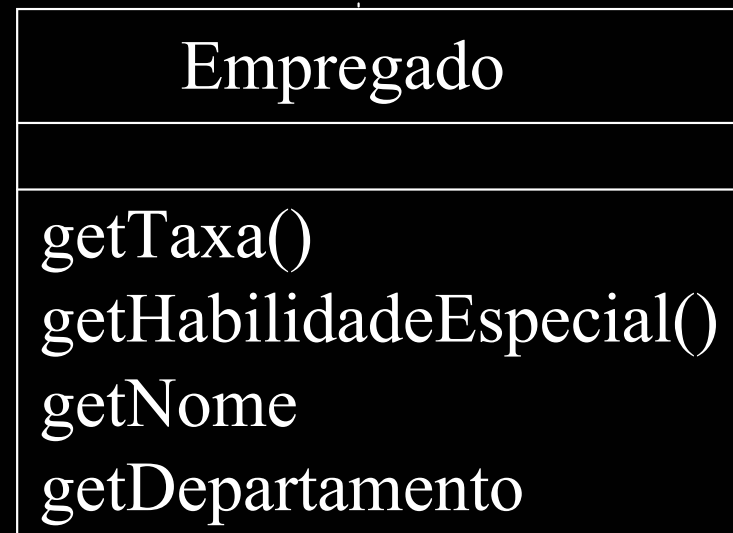
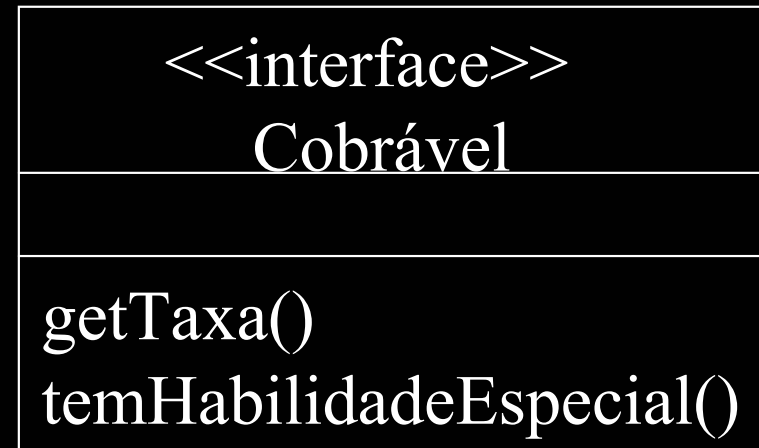
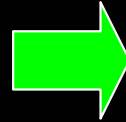
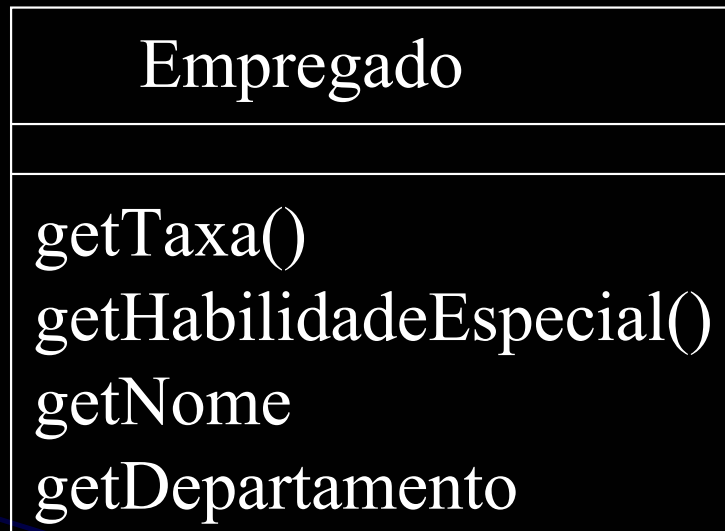
# Extrair Subclasse



# Extrair Superclasse

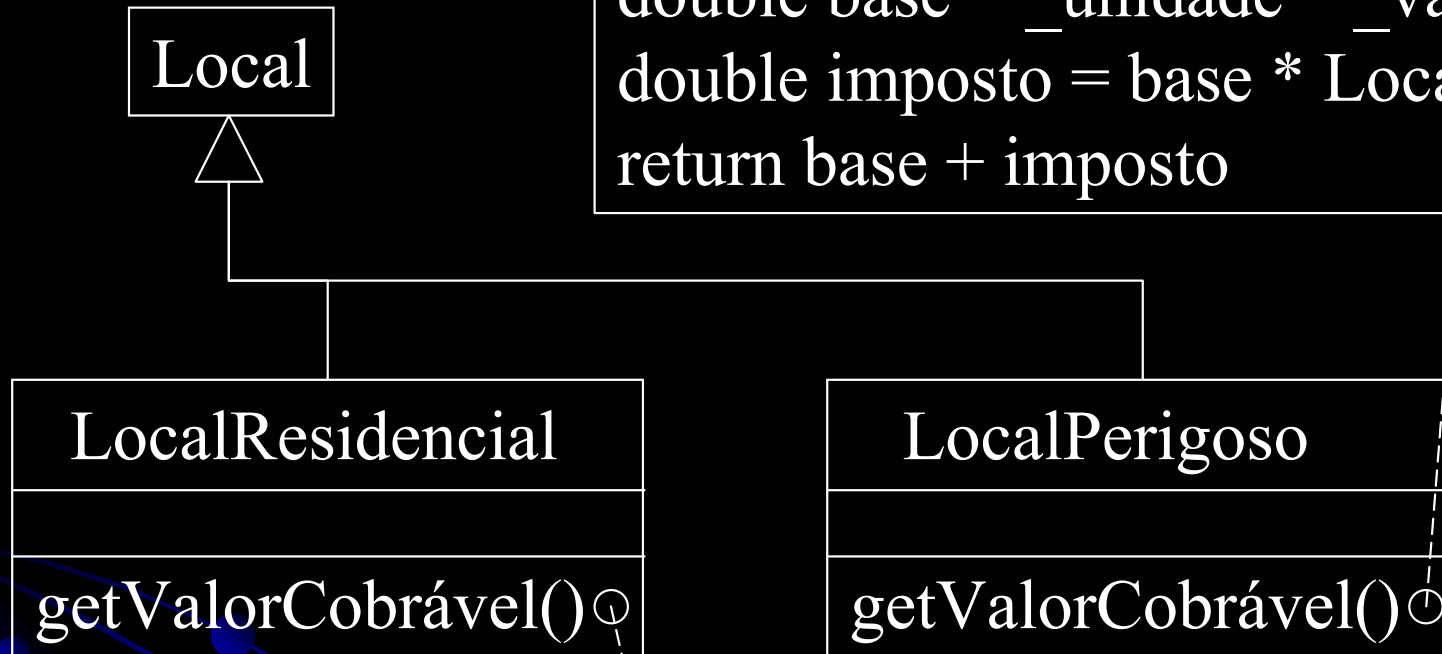


# Extrair Interface



# Criar um Método Padrão (Template Method)

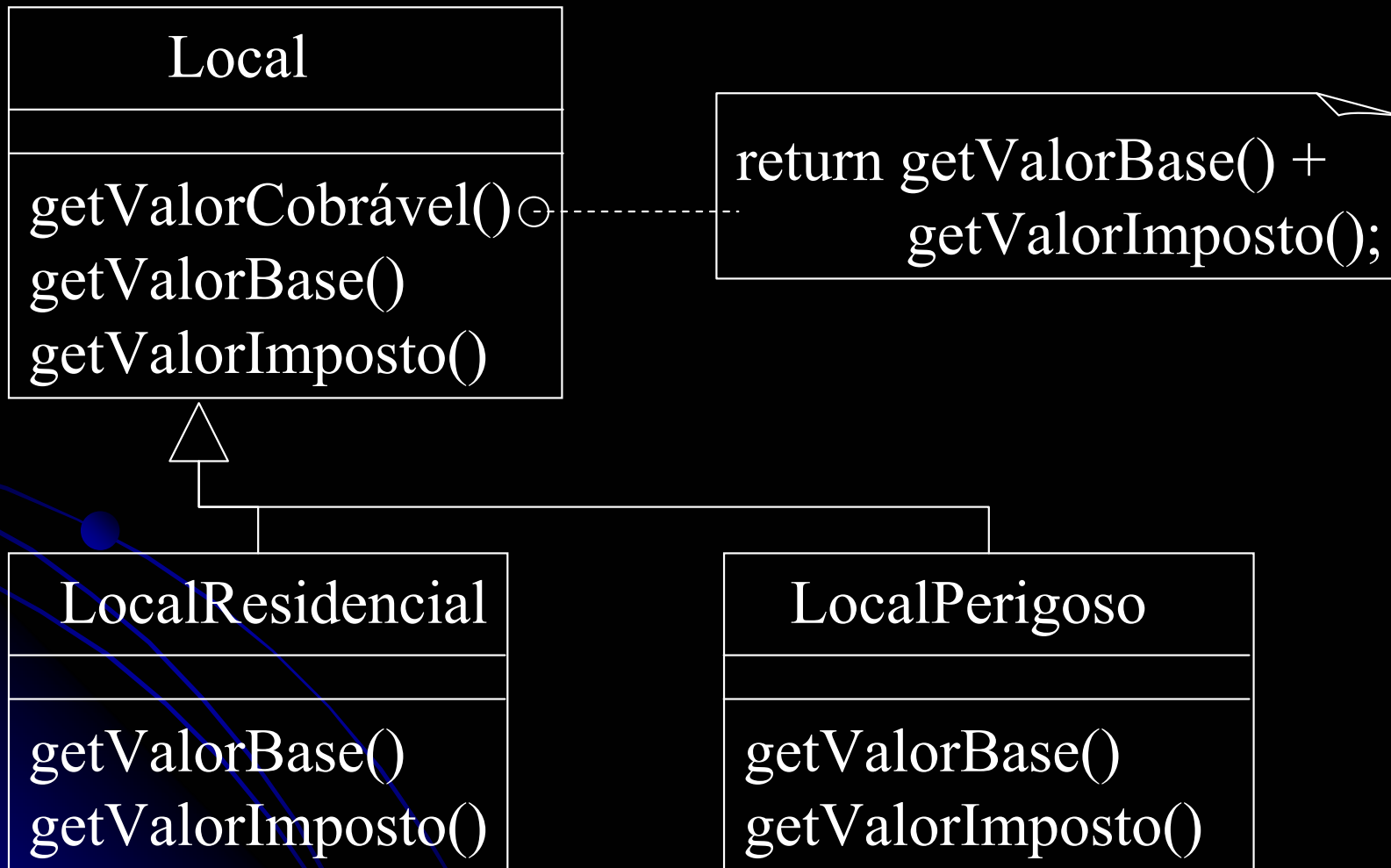
```
double base = _unidade * _valorUnitário * 0.5;  
double imposto = base * Local.TAXA * 0.2;  
return base + imposto
```



```
double base = _unidade * _valorUnitário;  
double imposto = base * Local.TAXA;  
return base + imposto
```



# Criar um Método Padrão (2)



# Mecânica

- Decomponha os métodos de modo que todos os métodos extraídos sejam idênticos ou completamente diferentes
- Use Subir Método na Hierarquia para subir os métodos idênticos
- Para os métodos diferentes, use Renomear Método de modo que as assinaturas para todos os métodos sejam as mesmas
- Compile e Teste após cada alteração



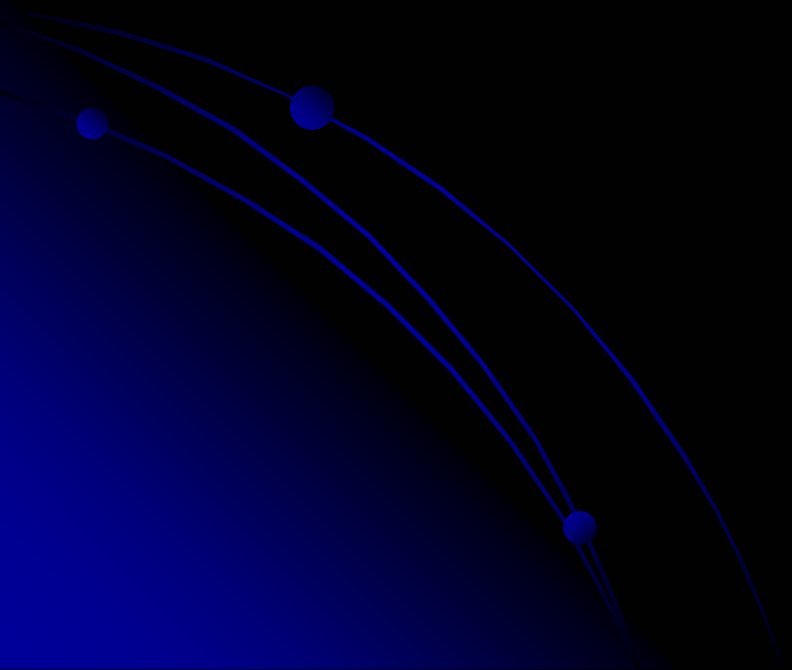
# Mecânica (2)

- Use Subir Método na Hierarquia em um dos métodos originais. Defina as assinaturas dos métodos diferentes na superclasse como métodos abstratos
- Compile e Teste
- Remova os outros métodos, compile e teste após cada remoção

# Resumo

- Temos visto a mecânica de várias refatorações
- Quando você detecta um mau cheiro, você frequentemente aplica refatoração para limpar o código
- Refatorações frequentemente levam a Padrões de Projeto

# Refatoração com Eclipse



# Refatoração com Eclipse

- Provê Refatorações primitivas automáticas poderosas que são a base para grandes Refatorações
- Move, Extract, Change Method Signature, Convert Anonymous Class to Nested, Convert Nested Type, Convert Local Variable to Field, Encapsulate Field, Decompose Conditional, Push Up, Pull Down, Rename e outros...
- Oferece Undo

# Refatorações Automáticas

- Refatoração à mão consome tempo
- Com ferramentas, refatoração se torna cada vez menos uma atividade separada da programação
- Erros de projeto se tornam menos caros
- Muito menos testes
- Porém, sempre haverá refatorações que não podem ser automatizadas

# Refatorações Grandes

- Acúmulo de vários problemas sobre o tempo pode corromper o projeto original
- Você não entende mais o sistema
- Acúmulo de decisões de projeto entendidas pela metade estrangula o programa

# Quatro Refatorações Grandes

- Desembaraçar Herança
- Converter Projeto Procedural em Objetos
- Separar o Domínio da Apresentação
- Extrair Hierarquia

# Resumo

- Tipicamente, Refatoração é feito em pequenos passos
- Depois de cada passo, você está com um sistema que preserva o comportamento original
  - Porém, tipicamente se mistura Refatoração com fixar bugs
- Refatoração pode ser usada para
  - entender o código
  - limpar o código
  - preparar o código para extensões



# Exercício 2

- Usar o Eclipse para implementar as refatorações vistas no exemplo da Conta dos filmes alugados

# Exercício 3

- Refatorar o programa para facilitar alterações na classificação dos filmes