Simplificando expressões condicionais

Expressões condicionais podem ser difíceis de serem entendidas na medida em que vão se tornando mais elaboradas. Há um grupo de refatorações capazes de lidar com essa complexidade de expressões condicionais.

Brevemente pode-se descrever o propósito de cada operação como sendo:

- Decompor condicional: quebrar a condicional em pedaços menores;
- Consolidar expressão condicional: quando existem vários testes e todos têm o mesmo efeito;
- Consolidar fragmentos de condicional duplicados: para remover duplicações em estruturas condicionais;
- Remover flag de controle: para deixar mais clara a estrutura de controle;
- Remover condicionais aninhadas com clausulas de guarda: idem à anterior, mas para estruturas condicionais aninhadas;
- Substituir condicional por polimorfismo: para expressar alterações de comportamento por meio de objetos;
- Introduzir objeto nulo: para remover a verificação de objetos nulos em código;
- Introduzir asserção (ou afirmativa): quando um trecho de código assume um determinado estado para que possa ser executado.

Obs.: As refatorações cujos nomes estão sublinhados não são apresentadas neste material. Para maiores informações consulte o livro adotado pela disciplina ou o site www.refactoring.com.

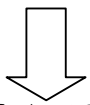
Refatorações do grupo "Simplificando Expressões Condicionais":

Decompor condicional

Situação: Você tem uma condicional complicada (IF- then- else)

Solução: Extraia métodos da condição, das partes **then,** e das partes else.

```
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);
```

Motivação:

À medida em que você escreve código para testar condções e realiza várias coisas dependentes dessas condições, rapidamente você se depara com um código longo (algo que o torna mais difícil de ler e entendê-lo).

Você pode tornar sua intenção mais clara ao decompor o método e substituir pedaços de código por uma chamada de métodos nomeada de acordo com a intenção daquele trecho. Com condições você pode receber mais benefícios se fizer isso com as partes **then** e **else** da expressão. Desse modo você deixa explícita e de fácil entendimento a condição e as partes que as compõe.

Mecânica da refatoração:

- Extraia a condição em seu próprio método.
- Extraia as partes **then** e **else** em seus próprios métodos.

Se encontrar condicionais aninhadas, inicialmente deve-se verificar se a refatoração "Substituir Condicionais aninhadas por cláusulas-guardas". Caso não seja possível, decomponha cada uma das condicionais.

Exemplo

Suponha o cálculo de algum valor, dependente da estação do ano (inverno ou verão):

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

Extraindo métodos da condicional e de cada uma das ramificações da condicional, o código torna:

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

private boolean notSummer(Date date) {
    return date.before (SUMMER_START) || date.after(SUMMER_END);
}

private double summerCharge(int quantity) {
    return quantity * _summerRate;
}

private double winterCharge(int quantity) {
    return quantity * _winterRate + _winterServiceCharge;
}
```

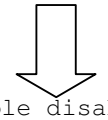
Aqui está apresentado o código ao final da refatoração. Contudo a refatoração deve ser conduzida em pequenos passos, um por vez.

Consolidar Expressão Condicional

Situação: Você possui uma seqüência de testes condicionais com o mesmo resultado.

Solução: Combine-os em uma única expressão condicional e extraia-a.

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



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

Motivação:

É comum encontrar uma série de testes condicionais em que cada teste é diferente mas os resultados são os mesmos. Quando se deparar com essa situação deve-se usar conectores lógicos **and's** e **or's** para agrupá-los em um único teste condicional com um único resultado.

Agrupar os testes condicionais é importante por duas razões:

- Ele torna o teste mais claro por mostrar que você está fazendo uma série de testes que em uma seqüência que possui o mesmo efeito.
- Geralmente permite a você usar "extrair método" (essa última refatoração sempre leva a código mais fácil de ser lido).

Contra-indicações para essa refatoração: quando os testes são de fato independentes e você julgar que eles não devem ser pensados em um conjunto (seu código já comunica bem sua intenção).

Mecânica da refatoração:

- Verifique se nenhuma das condicionais possui efeitos colaterais.

  Caso haja efeitos colaterais abandone essa refatoração.

- Substitua a string de condicionais por uma única expressão condicional utilizando operadores lógicos.
- Compile e teste.
- Avalie a utilização de "Extrair método" na condição.

Exemplo: OR's

Considere o exemplo mostrado na refatoração, basta que uma das condições seja verdadeira para que o cálculo não seja realizado.

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

A condicional pode ser rescrita da seguinte forma e apresenta o mesmo resultado anterior:

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

Por fim é possível extrair o código da condicional e usar uma chamada de métodos em seu lugar:

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

boolean isNotEligibleForDisability() {
    return ((_seniority < 2) || (_monthsDisabled > 12) ||
(_isPartTime));
 }
```

Exemplo AND's:

Seja o seguinte trecho de código:

```
if (onVacation())
    if (lengthOfService() > 10)
        return 1;
return 0.5;
```

Ele pode ser alterado para:

```
if (onVacation() && lengthOfService() > 10) return 1;
else return 0.5;
```

Por fim, pode-se ainda substituí-lo por um operador ternário:

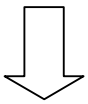
```
return (onVacation() && lengthOfService() > 10) ? 1 : 0.5;
```

Consolidar fragmentos de condicional duplicados

Situação: O mesmo fragmento de código está presente em todos os ramos de uma expressão condicional.

Solução: Mova-o para fora da expressão condicional.

```
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();
```

Motivação:

É comum encontrar o mesmo código sendo executado em todos os ramos de uma expressão condicional. Nesse caso você deve mover o código para fora da condicional, o que deixa mais claro quais as partes variáveis e quais as partes constantes do código.

Mecânica da refatoração:

- Identifique o código que é executado da mesma maneira, independente da condicional.
- Se o código em comum está no começo, mova-o para antes da condicional.
- Se o código em comum está no final, mova-o para depois da condicional.
- Se o código em comum está no meio, verifique se o código anterior ou posterior a ele difere em alguma coisa. Se ele mudar alterar algo você pode mover o código comum para frente ou para trás até os extremos do código. Você pode então movê-lo como está para fora do método, antes ou depois da condicional.
- Se há mais do que uma simples operação no código, você deve extrair seu código para um método.

Exemplo

Considerando o exemplo inicial dessa refatoração:

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

Como o método **send** é executado em ambos os casos, ele pode ser movido para fora da condicional. No caso, depois dela:

```
if (isSpecialDeal())
    total = price * 0.95;
```

```
else
    total = price * 0.98;
send();
```

O mesmo ocorre para tratamento de exceção: se um trecho de código aparece em vários tratamentos de exceção diferentes, ele pode ser movido para o bloco **final**.

Remover Flag de Controle

Situação: Você possui uma variável que está agindo como uma variável de controle para uma série de expressões booleanas.

Solução: Utilize **break** ou **return** em seu lugar.

Motivação: É comum em uma série de expressões condicionais encontrar *flags* de controle que determinam quando a estrutura condicional deve ser interrompida, algo do tipo:

```
set done to false
while not done
  if (condition)
      do something
      set done to true
  next step of loop
```

Tais estruturas condicionais são mais trabalhosas do que se imagina. Elas vêm de regras de programação estrutura em que chamadas a rotinas possuem uma única entrada e um único ponto de saída. Um ponto de saída leva o método a uma série de condicionais "misturadas" que dificultam muito o entendimento do projeto. Portanto, sempre que possível, liberte-se de flags de controle, substituindo-as por expressões condicionais muito mais claras.

Mecânica da refatoração:

(Para linguagens com instruções *break* e *continue*)

- Encontre o valor da flag de controle que faz com que a seqüência lógica seja encerrada.
- Substitua atribuições do valor de saída da expressão condicional por uma instrução **break** ou **continue.**
- Compile e teste após cada alteração.

(Para linguagens que não possuem as instruções *break* e *continue*)

- Extraia a lógica em um método.
- Encontre o valor da flag de controle que realiza a interrupção da seqüência lógica.
- Substitua atribuições do valor de saída da expressão condicional por uma instrução **return.**
- Compile e teste depois de cada modificação.

Exemplo

Substituindo flag de controle por *break*. Esse trecho de código emite um alerta quando uma seqüência de caracteres suspeita é encontrada:

```
void checkSecurity(String[] people) {
    boolean found = false;
    for (int i = 0; i < people.length; i++) {
        if (! found) {
           if (people[i].equals ("Don")){
             sendAlert();
             found = true;
           }
           if (people[i].equals ("John")){
             sendAlert();
             found = true;
           }
        }
    }
 }
```

Substituindo cada uma das atribuições à variável de controle por break, leva o código ao seguinte estado:

```
void checkSecurity(String[] people) {
    boolean found = false;
    for (int i = 0; i < people.length; i++) {
        if (! found) {
           if (people[i].equals ("Don")){
             sendAlert();
             break;
           }
           if (people[i].equals ("John")){
             sendAlert();
             break;
           }
        }
    }
 }
```

Neste ponto é possível remover todas as referências à variável de controle:

```
void checkSecurity(String[] people) {
    for (int i = 0; i < people.length; i++) {
        if (people[i].equals ("Don")){
           sendAlert();
           break;
        }
        if (people[i].equals ("John")){
           sendAlert();
           break;
        }
    }
 }
```

Exemplo 2:

Usando **return** como resultado de uma flag de controle. Neste caso o valor encontrado é retornado como resultado.

```
void checkSecurity(String[] people) {
    String found = "";
    for (int i = 0; i < people.length; i++) {
        if (found.equals("")) {
           if (people[i].equals ("Don")){
             sendAlert();
             found = "Don";
           }
           if (people[i].equals ("John")){
             sendAlert();
             found = "John";
           }
        }
    }
    someLaterCode(found);
}
```

Neste ponto é possível extrair todo o código que utiliza a variável de controle para um método à parte:

```
void checkSecurity(String[] people) {
    String found = foundMiscreant(people);
    someLaterCode(found);
}

String foundMiscreant(String[] people){
    String found = "";
    for (int i = 0; i < people.length; i++) {
        if (found.equals("")) {
           if (people[i].equals ("Don")){
             sendAlert();
             found = "Don";
           }
           if (people[i].equals ("John")){
             sendAlert();
             found = "John";
           }
        }
    }
    return found;
}
```

A flag de controle no novo método pode ser eliminada se retornar o valor diretamente:

```
String foundMiscreant(String[] people){
    for (int i = 0; i < people.length; i++) {
        if (people[i].equals ("Don")){
           sendAlert();
           return "Don";
        }
        if (people[i].equals ("John")){
           sendAlert();
           return "John";
        }
```