Decisões

Roland Teodorowitsch

Fundamentos de Programação - Escola Politécnica - PUCRS

24 de agosto de 2022

Introdução

Objetivos

- Implementar decisões (simples e complexas) usando o comando `if`
- Comparar números inteiros e de ponto-flutuante, e cadeias de caracteres
- Escrever comandos usando o tipo de dado `boolean`
- Desenvolver estratégias para testar seus programas
- Validar a entrada do usuário

Conteúdos

- O Comando `if`
- Comparando Números e *Strings*
- Múltiplas Alternativas
- Decisões Aninhadas
- Solução de Problemas: Fluxogramas
- Solução de Problemas: Casos de Teste
- Variáveis e operadores Booleanos
- Aplicação: Validação da Entrada

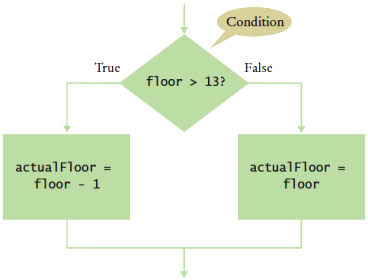
O Comando if

O Comando `if`

- Um programa de computador frequentemente necessita tomar decisões baseadas em alguma entrada ou circunstância
- Por exemplo, nos EUA, prédios frequentemente pulam o 13°andar, e os elevadores devem lidar com isto
  - O 14°andar é, na verdade, o 13°andar
  - Se $andar > 13$, então $andar = andar - 1$
- as duas palavras-chaves para o comando `if` são:
  - `if`
  - `else`
- o comando `if` permite que o programa execute ações diferentes conforme os valores das entradas ou dos dados que estão sendo processados

Fluxograma do comando `if`

- Conforme o resultado do teste, um dos dois ramos do `if` é executado uma vez
  - Ramo para teste verdadeiro/*true* (`if`) ou
  - Ramo para teste falso/*false* (`else`)

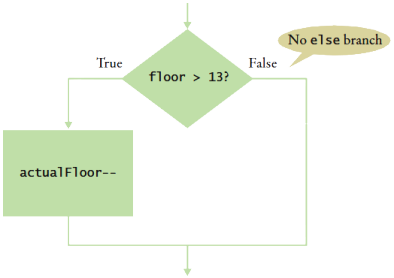



```
int actualFloor;

if (floor > 13)
{
   actualFloor = floor - 1;
}
else
{
   actualFloor = floor;
}
```

Fluxograma do comando `if` (sem `else`)

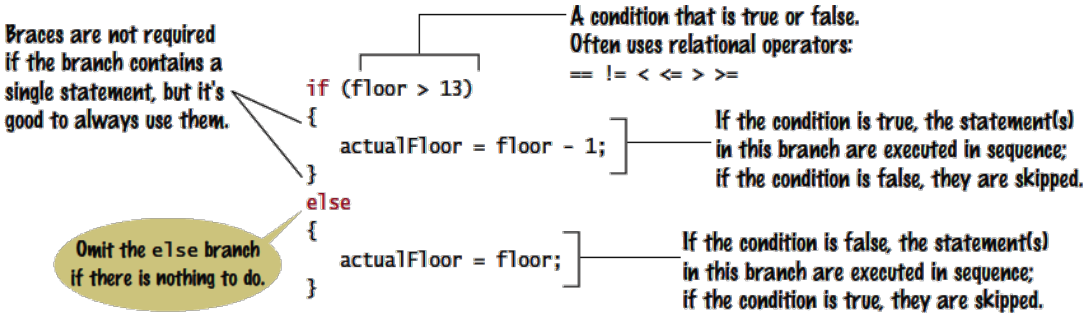
- Um comando `if` pode não necessitar do ramo para teste falso/*false* (`else`)




```
int actualFloor = floor;

if (floor > 13)
{
   actualFloor--;
} // No else needed
```

Sintaxe do Comando `if`



ElevatorSimulation.java (HORSTMANN, 2013, p. 84-85)

```java
import java.util.Scanner;

/**
   This program simulates an elevator panel that skips the 13th floor.
*/
public class ElevatorSimulation {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.print("Floor: ");
      int floor = in.nextInt();

      // Adjust floor if necessary
      int actualFloor;
      if (floor > 13) {
         actualFloor = floor - 1;
      }
      else {
         actualFloor = floor;
      }
      System.out.println("The elevator will travel to the actual floor " + actualFloor);
   }
}
```

Dicas sobre o Uso de Chaves

- É uma boa prática alinhar todos os pares de chaves verticalmente
  - Alinhados

```
if (floor > 13)
{
    floor--;
}
```

  - Não alinhados (economizando linhas)

```
if (floor > 13) {
    floor--;
}
```

Dicas sobre o Uso de Chaves

- Mesmo que para seleção de um único comando não seja necessário, sempre use chaves
  - Em vez de

```
if (floor > 13)
   floor--;
```

  - Prefira usar

```
if (floor > 13)
{
   floor--;
}
```

Dicas sobre Blocos Indentados

- O código Java é estruturado em blocos
- Use a tecla `Tab` para criar uma indentação consistente com número adequado de espaços
- Uma indentação consistente torna o entendimento do código muito mais fácil para humanos

```
public class ElevatorSimulation
{
|  public static void main(String[] args)
|  {
|  |  int floor;
|  |  . . .
|  |  if (floor > 13)
|  |  {
|  |  |  floor--;
|  |  }
|  |  . . .
|  }
|  |  |  |
0  1  2  3    Indentation level
```

Erros Comuns

- Um erro comum é colocar um `;` depois do comando `if`:

```
if (floor > 13) ;
{
    floor--;
}
```

- Um `;`, sem comando antes, é um **comando vazio**
- Portanto, no exemplo acima, o comando `if` **NÃO FARÁ NADA** se o teste for verdadeiro (comando vazio) e o bloco (entre chaves) será executado independentemente do teste

Comparando Números e Strings

Comparando Números e *Strings*

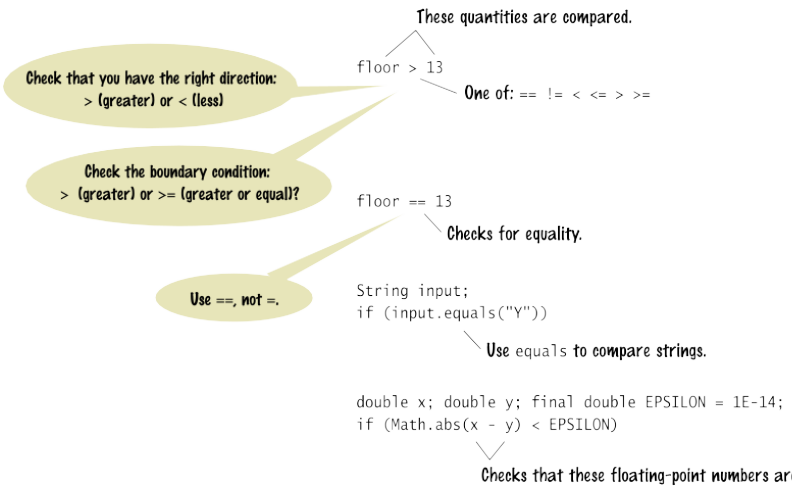
- Todo comando `if` tem uma condição que geralmente compara dois valores usando um operador

```
if (floor > 13) ...
if (floor >= 13) ...
if (floor < 13) ...
if (floor <= 13) ...
if (floor == 13) ...
if (floor != 13) ...
```

Operadores Relacionais

Java	Notação Matemática	Descrição
>	$>$	Maior que
>=	$\geq$	Maior ou igual que
<	$<$	Menor que
<=	$\leq$	Menor ou igual que
==	$=$	Igual
!=	$\neq$	Diferente

Sintaxe de Comparações



Precedência de Operadores

- Os operadores de comparação tem menor precedência do que operadores aritméticos
  - Cálculos são feitos antes das comparações
  - Normalmente os cálculos estão do lado direito de operadores de comparação ou atribuição

```
actualFloor = floor + 1;
if (floor > height + 1) ...
```

Exemplos de Operadores Relacionais (1)

Expressão	Valor	Comentário
`3 <= 4`	`true`	`3` é menor do que `4`; `<=` testa se é "menor ou igual".
`3 =< 4`	**X**	O operador "menor ou igual" é `<=`, e não `=<`. O símbolo "menor que" vem antes.
`3 > 4`	`false`	`>` é o oposto de `<=`
`4 < 4`	`false`	O lado esquerdo da comparação tem que ser menor do que o lado direito.
`4 <= 4`	`true`	Ambos os lados são iguais; `<=` testa se é "menor ou igual"
`3 == 5 - 2`	`true`	`==` testa igualdade.
`3 != 5 - 1`	`true`	`!=` testa diferença. É verdadeiro que `3` não é `5-1`.

Uso de Operadores Relacionais (2)

Expressão	Valor	Comentário
`3 = 6 / 2`	X	Use == para testar igualdade.
`1.0 / 3.0 == 0.333333333`	`false`	Embora os valores sejam muito próximos um do outro, eles não são exatamente iguais.
`"10" > 5`	X	Você não pode comparar um *string* com um número.
`"Tomate".substring(0,3).equals("Tom")`	`true`	Sempre use o método `equals` para verificar se duas *strings* tem os mesmos conteúdos.
`"Tomate".substring(0,3) == ("Tom")`	`false`	Nunca use == para comparar *strings*; este operador apenas verifica se os *strings* estão armazenados no mesmo local.

O Operador Condicional

- Há uma forma abreviada de seleção que não é usada no livro-texto, mas que poderá aparecer em outros códigos
- Trata-se de uma atribuição com a seguinte sintaxe:

```
variável = condição ? expressão_para_condição_verdadeira : expressão_para_condição_falsa ;
```

- Por exemplo:

```
actualFloor = floor > 13 ? floor - 1 : floor;
```

- Nesta forma abreviada aparecem todas as partes de um `if-else`, porém usando:
  - `?` para inciar o ramo verdadeiro e
  - `:` para iniciar o ramo falso

Comparando *Strings*

- *Strings* são um pouco “especiais” em Java
- Não use o operador == com *strings*
  - O seguinte trecho compara a localização de duas *strings* e não os seus conteúdos

```
if (string1 == string2) ...
```

  - Em vez disto uso o método `equals`:

```
if (string1.equals(string2)) ...
```

Erro Comum: Comparação de Números de Ponto-flutuante

- Números de ponto-flutuante tem precisão limitada
- Erros de arredondamento podem levar a resultados inesperados

```
double r = Math.sqrt(2.0);
if (r*r == 2.0) {
   System.out.println("Math.sqrt(2.0) ao quadrado eh 2.0");
}
else {
   System.out.println("Math.sqrt(2.0) ao quadrado "+
                      "nao eh 2.0, mas "+r*r);
}
```

- Resultado:

```
Math.sqrt(2.0) ao quadrado nao eh 2.0, mas 2.00000000000000044
```

O Uso de EPSILON

- Use um valor bastante pequeno para comparar se a diferença entre valores de ponto-flutuante está “suficientemente perto”
  - A magnitude da diferença entre os dois valores deve ser menor do que determinado limite
  - Matematicamente, diz-se que x e y estão suficientemente próximos se $|x - y| < \varepsilon$

```
final double EPSILON = 1E-14;
double r = Math.sqrt(2.0);
if (Math.abs(r * r - 2.0) < EPSILON) {
   System.out.println("Math.sqrt(2.0) ao quadrado eh "+
                      "aproximadamente 2.0");
}
```

Erro Comum: Usar == para Comparar *Strings*

- == compara a localização de *strings* e não o seu conteúdo
- Java cria uma nova variável `String` cada vez que é usado um texto entre aspas
  - Se há uma *string* que coincida exatamente com ela, Java a reusa

```
String nickname = "Rob";
...
if (nickname == "Rob") // O teste eh verdadeiro/true
```

```
String name = "Robert";
String nickname = name.substring(0,3);
...
if (nickname == "Rob") // O teste eh falso/false
```

Ordem Lexicográfica

- Para comparar *strings* pela ordem de dicionário, pode-se usar `compareTo()`
- Quando se compara `string1` usando `compareTo()` com `string2`, `string1` vem:
  - Antes de `string2`, se `string1.compareTo(string2) < 0`
  - Após `string2`, se `string1.compareTo(string2) > 0`
  - Na mesma posição que `string2`, se `string1.compareTo(string2) == 0`
- Observações:
  - Todas as letras maiúsculas vem antes das minúsculas
  - “Espaço” vem antes de todos os caracteres imprimíveis
  - Dígitos (0-9) vem antes das letras

Implementando um Comando `if`

- **Problema:** A livraria da Universidade realiza um Dia Kilobyte de Descontos sempre no dia 24 de outubro, dando um desconto de 8% em todos as compras de acessórios de computador se o preço for menor do que R$128,00, e um desconto de 16% se o preço é no mínimo R$128,00.

Implementando um Comando `if`

- **Passos:**
  1. Decida qual será a condição para decisão:
     `preço original < 128?`
  2. Escreva o pseudocódigo para o ramo verdadeiro:
     `preço com desconto = 0,92 x preço original`
  3. Escreva o pseudocódigo para o ramo falso:
     `preço com desconto = 0,84 x preço original`
  4. Faça uma verificação dos operadores relacionais, testando-os com valores abaixo (127), igual (128) e acima (129)
  5. Remova a duplicação:
     `preço com desconto = ___ x preço original`
  6. Teste ambos os ramos:
     `preço com desconto = 0,92 x 100 = 92`
     `preço com desconto = 0,84 x 200 = 168`
  7. Escreva o código em Java

Exemplo Implementado

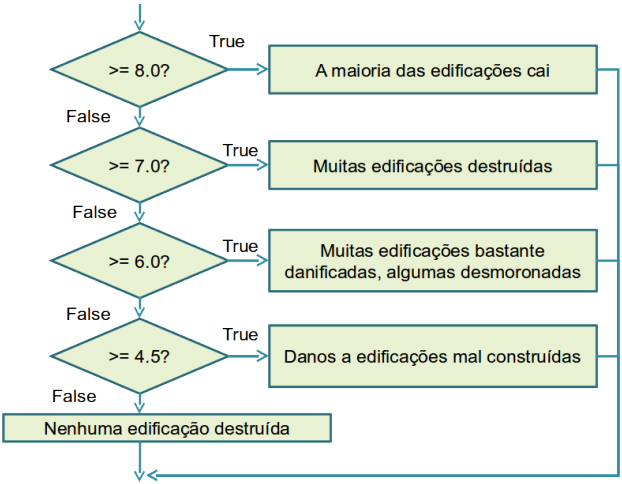
```
if (precoOriginal < 128)
{
   taxaDesconto = 0.92;
}
else
{
   taxaDesconto = 0.84;
}
precoComDesconto = taxaDesconto * precoOriginal;
```

Múltiplas Alternativas

Múltiplas Alternativas

- Um `if` tem 2 ramos, mas o que acontece se forem necessários mais do que dois ramos?
- Por exemplo, uma escala para o efeito de um terremoto:
  - 8 (ou mais): a maioria das edificações cai
  - 7 to 7.99: muitas edificações destruídas
  - 6 to 6.99: muitas edificações bastante danificadas, algumas desmoronadas
  - 4.5 to 5.99: danos a edificações mal construídas
  - menos do que 4.5: nenhuma edificação destruída

Fluxograma para Múltiplos Ramos



O que há de errado com este código?

```
if (richter >= 8.0) {
  System.out.println("A maioria das edificacoes cai");
}
if (richter >= 7.0) {
  System.out.println("Muitas edificacoes destruidas");
}
if (richter >= 6.0) {
  System.out.println("Muitas edificacoes bastante danificadas, "+
                     "algumas desmoronadas");
}
if (richter >= 4.5) {
  System.out.println("Danos a edificacoes mal construidas");
}
if (richter < 4.5) {
  System.out.println("Nenhuma edificacao destruida");
}
```

E com este código? O que há de errado?

```
if (richter >= 8.0) {
   System.out.println("A maioria das edificacoes cai");
}
if (richter < 8.0 && richter >= 7.0) {
   System.out.println("Muitas edificacoes destruidas");
}
if (richter < 7.0 && richter >= 6.0) {
   System.out.println("Muitas edificacoes bastante danificadas, "+
                      "algumas desmoronadas");
}
if (richter < 6.0 && richter >= 4.5) {
   System.out.println("Danos a edificacoes mal construidas");
}
if (richter < 4.5) {
   System.out.println("Nenhuma edificacao destruida");
}
```

Construção `if-else-if`

```
if (richter >= 8.0) {
   System.out.println("A maioria das edificacoes cai");
}
else if (richter >= 7.0) {
   System.out.println("Muitas edificacoes destruidas");
}
else if (richter >= 6.0) {
   System.out.println("Muitas edificacoes bastante danificadas, "+
                      "algumas desmoronadas");
}
else if (richter >= 4.5) {
   System.out.println("Danos a edificacoes mal construidas");
}
else {
   System.out.println("Nenhuma edificacao destruida");
}
```

Exercício

1. Escreva um programa em Java que leia dois valores reais, `x` e `y`, que correspondem às coordenadas de um ponto no plano cartesiano. E imprima em que quadrante este ponto se encontra (“1o. Quadrante”, “2o. Quadrante”, “3o. Quadrante”, “4o. Quadrante”) ou se ele se encontra na origem (“Origem”) ou sobre um dos eixos (“Eixo X”, “Eixo Y”).
   Em um primeiro momento use os comandos `if` e `else`. Depois reescreva o programa usando expressões condicionais.

Outra forma de Implementar Múltiplos Ramos

- O comando `switch` escolhe uma opção de um conjunto de opções
- Ele funciona com os tipos primitivos `byte`, `short`, `char` e `int`
- Também funciona com enumerações, a classe `String` e algumas outras classes (`Character`, `Byte`, `Short` e `Integer`) que podem ser convertidas para os tipos primitivos suportados
- O comando `break` encerra cada `case`
- `default` trata todas as opções não indicadas nos `case`s

Exemplo de `switch/case`

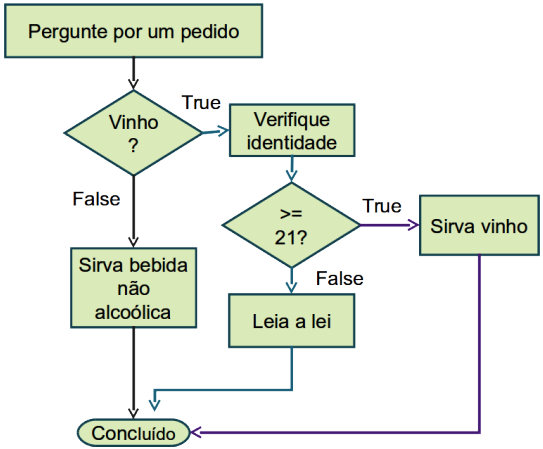
```
int digit = ...;
switch (digit) {
  case 1: digitName = "one";   break;
  case 2: digitName = "two";   break;
  case 3: digitName = "three"; break;
  case 4: digitName = "four";  break;
  case 5: digitName = "five";  break;
  case 6: digitName = "six";   break;
  case 7: digitName = "seven"; break;
  case 8: digitName = "eight"; break;
  case 9: digitName = "nine";  break;
  default: digitName = "";     break;
}
```

Decisões Aninhadas

Decisões Aninhadas

- Você pode aninhar um `if` dentro de um ramo de um comando `if`
- Um exemplo simples: pedidos de bebidas em um bar
  - Pergunte ao cliente o que ele quer beber
  - Se o cliente pedir vinho:
    - Peça a identidade do cliente
    - Se a idade do cliente for maior ou igual a 21, então
      Sirva vinho
    - Senão
      Polidamente explique como funciona a lei ao cliente
  - Senão
    - Sirva uma bebida não alcoólica ao cliente

Fluxograma de um `if` aninhado



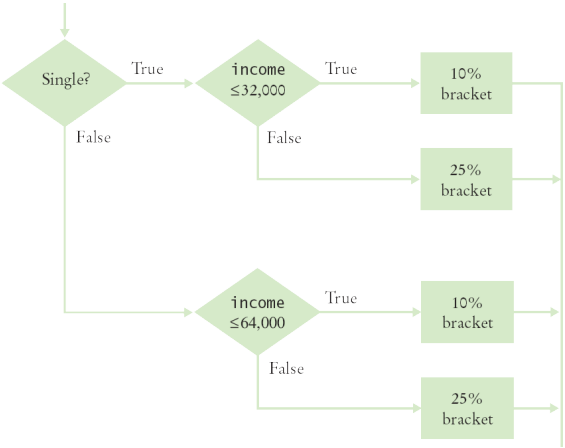
`if-else` aninhado dentro do ramo verdadeiro de um comando `if`: 3 partes

Exemplo de `if` aninhado: Taxas

O imposto de renda anual nos EUA considera basicamente 2 alternativas iniciais: contribuição para solteiros e contribuição para casados. Para cada uma destas possibilidades o imposto é calculado de forma diferente, conforme o ganho individual ou do casal.

- Solteiro
  - Renda menor ou igual a $32.000,00
    taxa = 10%
  - Renda maior do que $32.000,00
    taxa = $3.200,00 + 25% sobre o que exceder $32.000,00
- Casado
  - Renda conjunta menor ou igual a $64.000,00
    taxa = 10%
  - Renda conjunta maior do que $64.000,00
    taxa = $6.400,00 + 25% sobre o que exceder $64.000,00

Fluxograma para o Exemplo Taxas



`if-else` aninhado dentro dos ramos verdadeiro e falso de um comando `if`: 4 partes

TaxCalculator.java (Parte 1)

```java
import java.util.Scanner;

/**
  This program computes income taxes, using a simplified tax schedule.
 */
public class TaxCalculator {
   public static void main(String[] args) {
      final double RATE1 = 0.10;
      final double RATE2 = 0.25;
      final double RATE1_SINGLE_LIMIT = 32000;
      final double RATE1_MARRIED_LIMIT = 64000;
      double tax1 = 0;
      double tax2 = 0;

      // Read income and marital status
      Scanner in = new Scanner(System.in);
      System.out.print("Please enter your income: ");
      double income = in.nextDouble();
      System.out.print("Please enter s for single, m for married: ");
      String maritalStatus = in.next();
```

TaxCalculator.java (Parte 2)

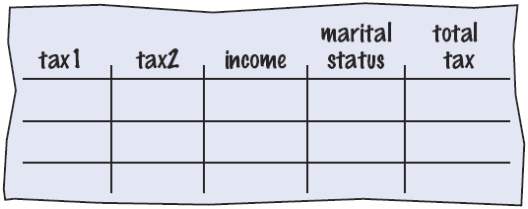
```
        // Compute taxes due
        if (maritalStatus.equals("s")) {
            if (income <= RATE1_SINGLE_LIMIT) {
                tax1 = RATE1 * income;
            }
            else {
                tax1 = RATE1 * RATE1_SINGLE_LIMIT;
                tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
            }
        }
        else {
            if (income <= RATE1_MARRIED_LIMIT) {
                tax1 = RATE1 * income;
            }
            else {
                tax1 = RATE1 * RATE1_MARRIED_LIMIT;
                tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
            }
        }
        double totalTax = tax1 + tax2;
        System.out.println("The tax is $" + totalTax);
    }
}
```

Acompanhamento Manual (ou Teste de Mesa)

- Acompanhar a execução manualmente ajuda a entender se o programa está funcionando corretamente ou não
- Crie uma tabela com as variáveis mais importantes
  - Use lápis e papel para acompanhar os valores destas variáveis
- Pode ser feito com pseudocódigo ou até mesmo código Java
  - Você pode marcar a execução no código com um *clips*
- Use valores de entrada:
  - Para os quais você já sabe os resultados que serão produzidos
  - Que testem todos os ramos possíveis do seu código

Teste de Mesa para o programa `TaxCalculator.java`

- Monte uma tabela para anotar os valores das variáveis



- Teste com as seguintes entradas
  - `income=20000, maritalStatus="s"`
  - `income=40000, maritalStatus="s"`
  - `income=40000, maritalStatus="m"`
  - `income=80000, maritalStatus="m"`

Erro Comum: `else` "desamparado"

- Quando um `if` é aninhado dentro de outro `if`, o seguinte pode ocorrer:

```
double shippingCharge = 5.00; // $5 inside continental U.S.
if (country.equals("USA"))
  if (state.equals("HI"))
    shippingCharge = 10.00;   // Hawaii is more expensive
else // Pitfall!
  shippingCharge = 20.00;     // As are foreign shipment
```

- O nível de indentação sugere que o `else` esteja relacionado ao `if` de `country.equals("USA")`
- Porém a cláusula `else` sempre se associa ao `if` mais próximo

Tipos Enumerados

Java provê uma forma simples de nomear uma lista finita de valores que uma variável pode armazenar

- Funciona como uma declaração de novos tipos, com uma lista de possíveis valores

```
public enum FilingStatus {
 SINGLE, MARRIED,MARRIED_FILING_SEPARATELY
}
```

- Você pode ter qualquer número de valores, mas tem que incluir eles todos na declaração `enum`
- Você pode declarar variáveis do tipo “enumeração”

```
FilingStatus status = FilingStatus.SINGLE;
```

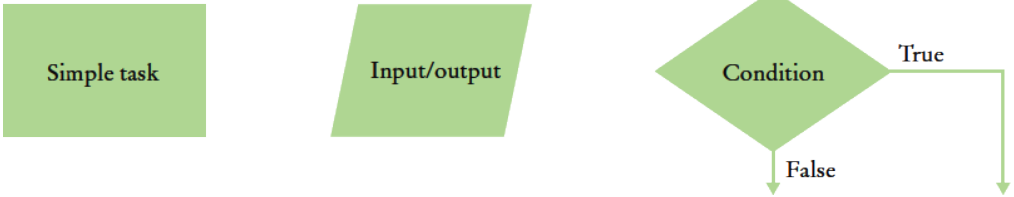
- Você também pode usar o operador de comparação entre eles:

```
if (status == FilingStatus.SINGLE) ...
```

Solução de Problemas: Fluxogramas

Solução de Problemas: Fluxogramas

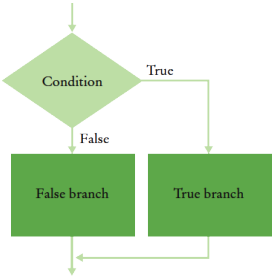
- Já vimos alguns exemplos de fluxogramas
- Um fluxograma mostra a estrutura de tarefas e decisões que tem que ser executadas para resolver um problema
- Os elementos básicos de um fluxograma são:



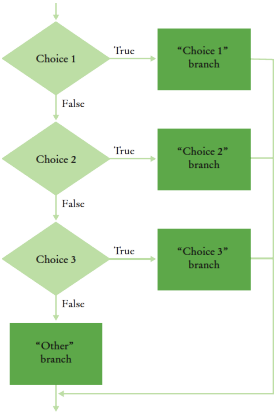
- Os elementos de um fluxograma são conectados com setas
- Nunca aponte uma seta para dentro de outro ramo (isto cria “código spaghetti”, que dificulta a manutenção)

Fluxogramas Condicionais

`if` com 2 resultados



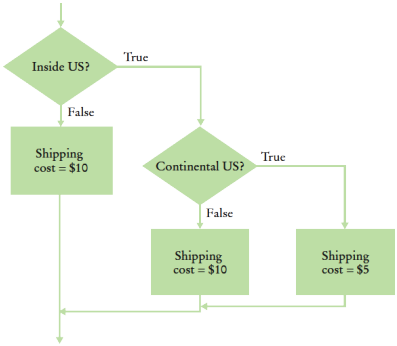
`if` com vários resultados



Fluxograma: Custos de Envio nos EUA

O custo de envio interno nos EUA é de $5, exceto para Hawaii e Alaska para onde o custo é de $10. O custo internacional é de $10.

- 3 ramos:



Solução de Problemas: Casos de Teste

Solução de Problemas: Casos de Teste

- Procure fazer uma cobertura completa de todos os pontos de decisão
  - Todos os ramos do seu código devem ser cobertos por um caso de teste
  - Por exemplo, há duas possibilidades para o estado civil e duas possibilidades de taxa, levando a 4 casos de teste
  - Teste vários valores de limite (valores que são testados no seu código), tais como uma renda que esteja entre as duas possibilidades, e uma renda igual a zero
  - Se você estiver responsável pela verificação de erro, também teste entradas inválidas, tais como rendas negativas

Casos de Teste para o Cálculo de Taxas

- Escolha valores de entrada que testem limites e valores nulos e também teste cada ramo do código

Caso de Teste	Saída Esperada	Comentário
30.000 s	3.000	Taxa de 10%
72.000 s	13.200	3.200 + 25% de 40.000
50.000 m	5.000	Taxa de 10%
104.000 m	16.400	6.400 + 25% de 40.000
32.000 m	3.200	Caso limite
0	0	Caso limite

Variáveis e operadores Booleanos

Variáveis e operadores Booleanos

- Variáveis Booleanas
  - Uma variável booleana é frequentemente chamada de *flag* porque pode assumir ou o valor verdadeiro (*true*) ou falso (*false*)
  - Java dispõe do tipo `boolean` para variáveis booleanas, que podem assumir ou `true` ou `false`

```
boolean acertou = true;
boolean sair = false;
```

- Operadores Booleanos: `&&` e `||`
  - São usados para combinar múltiplas condições
  - `&&` é o operador AND (E)
  - `||` é o operador OR (OU)

Condições Combinadas: &&

- A combinação de dois testes ou condições é usada frequentemente para verificar se um valor está dentro de um intervalo
- Ambos os lados do AND devem ser verdadeiros para que o resultado também seja:

```
if (temp > 0 && temp < 100) {
  System.out.println("Liquido");
}
```

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

Condições Combinadas: ||

- Se apenas um dos testes precisa ser verdadeiro, pode-se combinar os testes com OR
- Se um dos lados do OR for verdadeiro, o resultado também será:

```
if (balance > 100 || credit > 100) {
  System.out.println("Aceito");
}
```

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

O Operador NOT: !

- Se for necessário inverter o valor de uma variável booleana ou de uma condição, basta precedê-la com `!`:

A	!A
true	false
false	true

- O que é melhor?

```
if (!attending || grade < 60)
   System.out.println("Drop?");
```

```
if (attending && !(grade < 60))
   System.out.println("Stay");
```

- Ao usar `!`, procure simplificar a lógica:

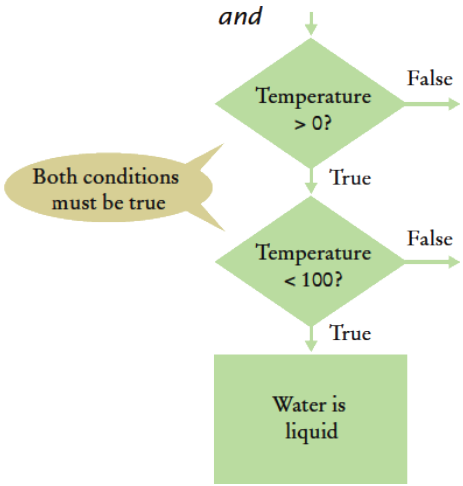
```
if (attending && grade >= 60) ...
```

Fluxograma para AND

- Isto é frequentemente chamado de “verificação de limite”, sendo usado para validar se uma entrada está entre 2 valores

```
if (temp > 0 && temp < 100) {
  System.out.println("Liquid");
}
```

Fluxograma para AND

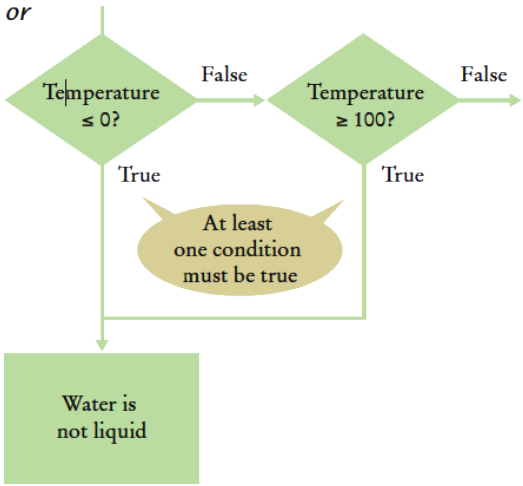


Fluxograma para OR

- Outra forma de “verificação de limite”: verifica se o valor está fora do limite

```
if (temp <= 0 || temp >= 100) {
  System.out.println("Not Liquid");
}
```

Fluxograma para OR



Exemplos de Uso de Operadores Booleanos (1)

Expressão	Valor	Comentário
`0 < 200 && 200 < 100`	`false`	Apenas a primeira condição é verdadeira.
`0 < 200 || 200 < 100`	`true`	A primeira condição é verdadeira.
`0 < 200 || 100 < 200`	`true`	O operador `||` não é um operador para "ou-ou". Se ambas as condições são verdadeiras, o resultado é verdadeiro.
`0 < x && x < 100 || x == -1`	`(0 < x && x < 100) || x == -1`	O operador `&&` tem maior precedência que o operador `||`.
`0 < x < 100`	**ERRO**	**Erro:** Esta expressão não testa se `x` está entre 0 e 100. A expressão `0 < x` gera um valor booleano, que não pode ser comparado com o valor inteiro 100.

Exemplos de Uso de Operadores Booleanos (2)

Expressão	Valor	Comentário
`x && y > 0`	**ERRO**	**Erro:** Esta expressão não testa se `x` e `y` são ambos positivos. A parte à direita de `&&` é um inteiro (`x`) e a parte direita é um booleando (`y > 0`). Não se pode usar `&&` com um valor inteiro.
`!(0 < 200)`	`false`	`0 < 200` é verdadeiro, portanto, a sua negação é falsa.
`frozen == true`	`frozen`	Não é necessário comparar uma variável booleana com `true`.
`frozen == false`	`!frozen`	É mais claro usar `!` do que comparar com `false`.

Erros Comuns (1)

- Combinação de múltiplos operadores relacionais
  - O seguinte formato é usado na matemática, mas não em Java:

```
if (0 <= temp <= 100)  // Erro de sintaxe!
```

  - São necessárias duas comparações:

```
if (0 <= temp && temp <= 100)
```

- Isto também não é permitido em Java:

```
if (input == 1 || 2)  // Erro de sintaxe!
```

  - Isto também exige 2 comparações:

```
if (input == 1 || input == 2)
```

Erros Comuns (2)

Confundir condições `&&` e `||`

- É um erro surpreendentemente comum confundir condições `&&` e `||`
- Se um valor está no intervalo de 0 a 100, ele é no mínimo 0 **e** no máximo 100
- Se ele está fora deste intervalo, ele é menor do que 0 **ou** maior do que 100
- Não há nenhuma regra de mágica; basta pensar com cuidado

Avaliação *short-circuit*: `&&`

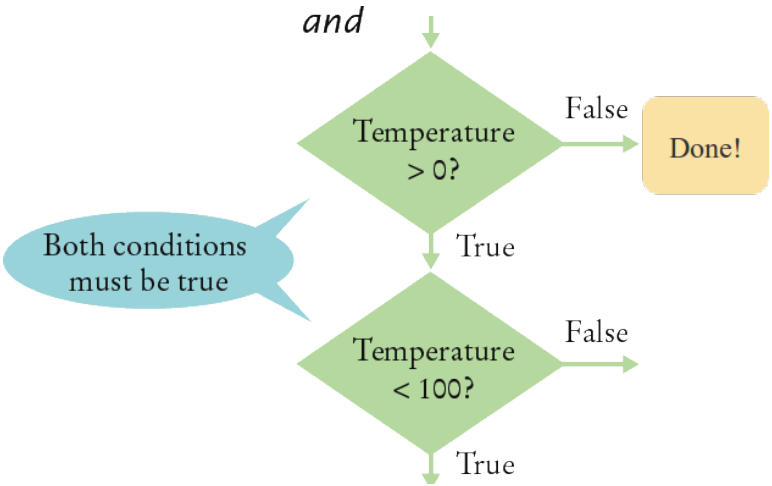
- Condições combinadas são avaliadas da esquerda para a direita
- Se uma das condições avaliadas for falsa, por que continuar avaliando as demais?

```
if (temp > 0 && temp < 100) {
  System.out.println("Liquid");
}
```

- Um exemplo útil:

```
if (quantity > 0 && price / quantity < 10)
```

Avaliação *short-circuit*: &&

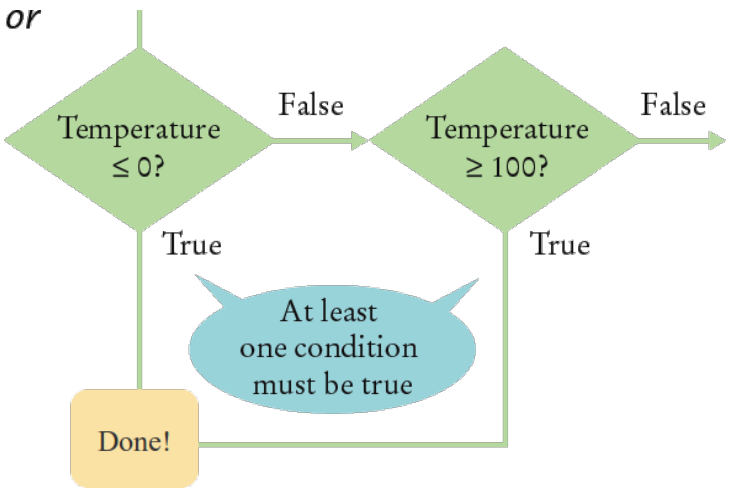


Avaliação *short-circuit*: ||

- Se alguma condição à esquerda for verdadeira, por que continuar avaliando as demais?

```
if (temp <= 0 || temp >= 100) {
  System.out.println("Not Liquid");
}
```

Avaliação *short-circuit*: | |



Lei de De Morgan

- A Lei de De Morgan diz como negar condições `&&` e `||`
  - `!(A && B)` é o mesmo que `!A || !B`
  - `!(A || B)` é o mesmo que `!A && !B`
- Exemplo: Envio para AK e HI é mais caro

```
if ( !(country.equals("USA") && !state.equals("AK") && !state.equals("HI")) )
   shippingCharge = 20.00;
```

```
if ( !country.equals("USA") || state.equals("AK") || state.equals("HI") )
   shippingCharge = 20.00;
```

- Para simplificar condições com negações de condições AND ou OR, geralmente é uma boa ideia aplicar a Lei de De Morgan para mover as negações para o nível mais interno.

Aplicação: Validação da Entrada

Aplicação: Validação da Entrada

- Aceitar entrada do usuário é perigoso
- Considere, por exemplo, o programa do elevador
  - O usuário pode fornecer um caracter ou valor inválido
  - Ele deveria fornecer um valor inteiro
  - O método `hasNextInt` da classe Scanner pode ajudar: ele retorna `true` se o valor for um inteiro válido, ou `false` em caso contrário
  - Depois pode-se verificar o intervalo do andar fornecido: deve estar entre 1 e 20; não deve ser 0, nem 13, nem > 20

```
if (in.hasNextInt()) {
  int floor = in.nextInt();
  // Process the input value
}
else {
  System.out.println("Not integer.");
}
```

ElevatorSimulation2.java (Parte 1)

```
import java.util.Scanner;

/**
   This program simulates an elevator panel that skips the 13th floor, checking for
   input errors.
 */
public class ElevatorSimulation2 {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.print("Floor: ");
      if (in.hasNextInt()) {
         // Now we know that the user entered an integer
         int floor = in.nextInt();
         if (floor == 13) {
            System.out.println("Error: There is no thirteenth floor.");
         }
         else if (floor <= 0 || floor > 20) {
            System.out.println("Error: The floor must be between 1 and 20.");
         }
         else {
```

ElevatorSimulation2.java (Parte 2)

```
            // Now we know that the input is valid
            int actualFloor = floor;
            if (floor > 13) {
               actualFloor = floor - 1;
            }
            System.out.println("The elevator will travel to the actual floor "
            + actualFloor);
         }
      }
      else {
         System.out.println("Error: Not an integer.");
      }
   }
}
```

Métodos para Teste de Caracteres

- A classe `Character` tem um bom número de métodos que retornam o tipo `boolean`:

```
if (Character.isDigit(ch)) {
  ...
}
```

Método	Exemplo de Caracteres Aceitos
`isDigit`	0, 1, 2
`isLetter`	A, B, C, a, b, c
`isUpperCase`	A, B, C
`isLowerCase`	a, b, c
`isWhiteSpace`	espaço, nova linha, tabulação

Exercícios

Exercícios (1)

1. Escreva um programa em Java que verifique se um número real é positivo, negativo ou zero.
2. Tendo como dados de entrada a altura e o sexo de uma pessoa, construa um programa em Java que calcule o seu peso ideal, utilizando as seguintes fórmulas:
   - Para mulheres: (62.1 * altura) - 44.7
   - Para homens: (72.7 * altura) - 58
3. Escreva um programa em Java que leia o ano de nascimento de uma pessoa, calcule e mostre sua idade, e também verifique e mostre:
   - se ela já deve votar (obrigatório para pessoas com idade entre 18 e 70 anos), pode votar (opcional para pessoas com idade entre 16 e 18 anos ou maior que 70 anos) ou não pode votar (impedido para pessoas com idade menor que 16 anos); e
   - se ela tem idade para conseguir Carteira de Habilitação (18 anos ou mais).

Exercícios (2)

4. Escreva um programa em Java que leia o código (inteiro) de um determinado produto e mostre a sua classificação. Utilize as seguintes categorias para classificar os produtos: 1 para “Alimento não perecível”; 2, 3 ou 4 para “Alimento perecível”; 5 ou 6 para “Vestuário”; 7 para “Higiene pessoal”; e qualquer outro código para “Inválido”.

Exercícios (3)

5. Escreva um programa em Java que leia as 4 notas de um aluno de Fundamentos de Programação (P_1, P_2, P_3 e M_T), calcule o seu grau G_1 e mostre este grau e uma mensagen indicando se o aluno passou por média ($G_1 \geq 7$), ficou em G_2 ($G_1 \geq 4$ e $G_1 < 7$) ou reprovou ($G_1 < 4$). Caso o aluno tenha ficado em G_2, calcule qual a nota mínima que o aluno deve tirar nesta prova para obter aprovação. Considere que:
   - M_T já corresponde à média calculada dos trabalhos e exercícios realizados.
   - A média de G_1 é calculada com a seguinte fórmula:

$$G_1 = \frac{P_1 + 2 \times P_2 + 3 \times P_3 + M_t}{7}$$

   - Para obter aprovação depois da realização do G_2, a média aritmética entre G_1 e G_2 deve ser maior ou igual a 5.

Exercícios (4)

6. Escreva um trecho de programa em Java para calcular o número de pontos e o valor de multas de trânsito por excesso de velocidade. Inicialmente seu programa deverá ler o limite de velocidade da via e a velocidade do veículo (medida com um radar), ambos em quilômetros por hora e sem casas decimais. Para determinar a velocidade do veículo que será considerada, aplica-se uma tolerância de 7 km/h, se a velocidade medida for menor ou igual a 100 km/h, ou de 7%, em caso contrário. Se a velocidade considerada for menor ou igual ao limite da via, não há multa. Se a velocidade considerada exceder 50% do limite da via, a multa será de R$880,41 (infração gravíssima, 7 pontos). Se a velocidade considerada exceder 20% do limite da via, a multa será de R$195,23 (infração grave, 5 pontos). Senão a multa será de R$130,16 (infração média, 4 pontos).

Referências

Referências

HORSTMANN, C. **Java for Everyone – Late Objetct**. 2. ed. Hoboken: Wiley, 2013. xxxiv, 589 p.