Instrução Java

Operadores condicionais ou relacionais de Java

Se você precisar alterar a execução do programa com base em uma determinada condição, poderá usar as instruções “if”. Os operadores relacionais determinam o relacionamento que um operando tem com o outro. Especificamente, eles determinam a condição de igualdade. Java fornece seis operadores relacionais, listados na tabela abaixo.

Fonte:Autoral

O resultado dessas operações é um valor booleano. Os operadores relacionais são usados com mais frequência nas expressões que controlam a instrução “if” e as várias instruções de loop. Qualquer tipo em Java, incluindo números inteiros, números de ponto flutuante, caracteres e booleanos, pode ser comparado usando o teste de igualdade, = =, e o teste de desigualdade, !=. Observe que em Java a igualdade é denotada com dois sinais de igual (“= =”), não um (“=”).

Em Java, a instrução mais simples que você pode usar para tomar uma decisão é a instrução if. Sua forma mais simples é mostrada aqui:

if(condition) instrução;

or

```
if (condition) instrução1;

else instrução 2;
```

Aqui, a condição é uma expressão booleana. Se a condição for verdadeira, a instrução será executada. Se a condição for falsa, a instrução será ignorada. Por exemplo, suponha que você tenha declarado uma variável inteira chamada algumaVariável e deseje imprimir uma mensagem quando o valor de algumaVariável for 10. O fluxograma e o código Java da operação são como abaixo,

Fonte:Autoral

```
if(algumaVariavel==10){
    System.out.println("O valor é 10”);
}
```

Podemos ter diferentes tipos de declarações “if”.

Blocos aninhados

Uma declaração if aninhada é uma instrução if que é o destino de outra iforelse. Em outros termos, podemos considerar uma ou várias instruções if dentro de um bloco if para verificar várias condições. Por exemplo, temos duas variáveis e queremos verificar uma condição específica para ambas, podemos usar blocos if aninhados.

Fonte: Autoral

if -else-if

Podemos ter uma situação em que precisamos verificar o valor várias vezes para encontrar a condição exata de correspondência. O programa abaixo explica a mesma coisa. Vamos ver que temos um requisito para verificar se o valor da variável é menor que 100, igual a 100 ou mais que 100. O código abaixo explica a mesma lógica usando a escada if-else-if.

Fonte: Autoral

Resultado:

Fonte: Autoral

Operadores lógicos de curto-circuito

O Java fornece dois operadores booleanos interessantes, não encontrados em muitas outras linguagens de computador. Essas são versões secundárias dos operadores booleanos AND e OR e são conhecidas como operadores lógicos de curto-circuito. Dois operadores lógicos de curto-circuito são os seguintes:

· && curto-circuito E

· || curto-circuito OU

Eles são usados para vincular pequenas expressões booleanas para formar expressões booleanas maiores. Os operadores && e ||

avaliam apenas valores booleanos. Para que uma expressão AND (&&) seja verdadeira, ambos os operandos devem ser verdadeiros. Por exemplo, a instrução abaixo é avaliada como verdadeira porque o operando um (2 <3) e o operando dois (3 <4) são avaliados como verdadeiros.

```
if ((2 < 3) && (3 < 4)) { }
```

O recurso de curto-circuito do operador && é assim chamado porque não perde tempo com avaliações inúteis. Um curto-circuito && avalia primeiro o lado esquerdo da operação (operando um) e, se for falso, o operador && não se preocupa em olhar para o lado direito da expressão (operando dois), pois o operador && já sabe que a expressão completa não pode ser verdadeira.

O operador || operador é semelhante ao operador &&, exceto que ele é avaliado como verdadeiro se QUALQUER dos operandos for verdadeiro. Se o primeiro operando em uma operação OR for verdadeiro, o resultado será verdadeiro, portanto, o curto-circuito || não perde tempo olhando para o lado direito da equação. Se o primeiro operando for falso, no entanto, o curto-circuito || precisa avaliar o segundo operando para ver se o resultado da operação OR será verdadeiro ou falso.

Fonte: Autoral

Resultado:

Fonte: Autoral

Operador Condicional Ternário

O operador condicional é um operador ternário (possui três operandos) e é usado para avaliar expressões booleanas, como uma instrução if, exceto em vez de executar um bloco de código se o teste for verdadeiro, um operador condicional atribuirá um valor a uma variável. Um operador condicional começa com uma operação booleana, seguida por dois valores possíveis para a variável à esquerda do operador de atribuição (=). O primeiro valor (aquele à esquerda dos dois pontos) é atribuído se o teste condicional (booleano) for verdadeiro e o segundo valor é atribuído se o teste condicional for falso. No exemplo abaixo, se a variável a for menor que b, o valor da variável x seria 50 ou se for maior, x = 60.

Fonte: Autoral

No exemplo abaixo, estamos decidindo o status com base na entrada do usuário, se aprovada ou falhou.

Saídas baseadas na entrada do usuário:

Fonte: Autoral

O operador NOT

Também chamado de complemento bit a bit, o operador NOT unário, ~, inverte todos os bits do seu operando. Se aplicado ao operando inteiro, ele reverterá todos os bits da mesma forma. Se aplicado ao literal booleano, ele será revertido.

int a = 23; // 23 é representado em binário como 10111

int b = ~ a; // isso reverte os bits 01000, que são 8 em decimal boolean x = true;

boolean y =! x; // Isso atribuirá valor falso a y, pois x é verdadeiro

O operador AND

O operador AND "&" produz 1 bit se ambos os operandos forem 1 caso contrário 0 bit. Da mesma forma, para operandos booleanos, resultará em true se ambos os operandos forem verdadeiros, caso contrário, o resultado será false.

int var1 = 23; // valor booleano seria 010111

int var2 = 33; // valor booleano seria 100001

int var3 = var1 & var2 // resulta no binário 000001 e no decimal 1 b1 boolean = true;

boolean b2 = false;

boolean b3 = b1 & b2; // b3 seria falso

O operador OR

O operador OR "|" produz um bit 0 se ambos os operandos forem 0, caso contrário, 1 bit. Da mesma forma, para operandos booleanos, resultará em false se ambos os operandos forem falsos ou o resultado será true.

int var1 = 23; // valor booleano seria 010111

int var2 = 33; // valor booleano seria 100001

int var3 = var1 | var2 // resulta no binário 110111 e no decimal 55 b1 boolean = true;

boolean b2 = false;

boolean b3 = b1 | b2; // b3 seria verdadeiro

O operador XOR (OU exclusivo)

O operador XOR "^" produz um bit 0 se ambos os operandos forem iguais (ambos 0 ou 1) caso contrário, 1 bit. Da mesma forma, para operandos booleanos, resultará em false se os dois operandos forem iguais (ambos são falsos ou ambos verdadeiros) ou o resultado será true.

int var1 = 23; // valor booleano seria 010111

int var2 = 33; // valor booleano seria 100001

int var3 = var1 ^ var2 // resulta no binário 110110 e no decimal 54 boolean b1 = true;

```
boolean b2 = false;

boolean b3 = b1 ^ b2; // b3 seria verdadeiro
```

Vejamos o programa abaixo, que demonstra os operadores acima para operações booleanas e inteiras.

Fonte: Autoral

Switch

Uma linguagem de programação usa instruções de controle para fazer com que o fluxo de execução avance e se ramifique com base nas alterações no estado de um programa. Java suporta duas instruções de controle de fluxo: if e switch. Estas instruções permitem controlar o fluxo da execução do seu programa com base nas condições conhecidas apenas durante o tempo de execução.

Instrução Switch

A instrução switch é a instrução de ramificação múltipla do Java. Ele fornece uma maneira fácil de despachar a execução para diferentes partes do seu código com base no valor de uma expressão. Aqui está a forma geral de uma instrução switch:

```
switch (expressão) {

case value1:
```

```
// sequência de instruções

break;

case value2:

// sequência de instruções

break;

...

casevalueN:

// sequência de instruções

break;

default:

// sequência de instruções padrão

}
```

A expressão deve ser do tipo byte, curto, int ou char; cada um dos valores especificados nas instruções de case deve ser de um tipo compatível com a expressão. Um valor de enumeração também pode ser usado para controlar uma instrução de opção. A partir do Java 7, a String também é permitida como expressão de case. Cada valor de case deve ser um literal exclusivo (ou seja, deve ser uma constante, não uma variável). Valores de case duplicados não são permitidos.

Como a instrução switch funciona

O valor da expressão é comparado com cada um dos valores literais nas instruções de case. Se uma correspondência for encontrada, a sequência de códigos após a instrução case é executada. Se nenhuma das constantes corresponder ao valor da expressão, a instrução padrão será executada. No entanto, a instrução padrão é opcional. Se nenhum caso corresponder e nenhum padrão estiver presente, nenhuma outra ação será tomada.

A instrução break é usada dentro do switch para finalizar uma sequência de instruções. Quando uma instrução break é encontrada, a execução ramifica para a primeira linha de código que segue toda a instrução switch.

Vamos entender o conceito do programa. No programa abaixo, imprima o valor do bônus com base no grau de funcionário. Um funcionário pode ser do tipo A, B, C ou padrão (qualquer coisa que não seja A, B & C).

Fonte: Autoral

Instruções do comutador aninhado

Você pode usar um comutador como parte da sequência de instruções de um comutador externo. Isso é chamado de switch aninhado. Como uma instrução switch define seu próprio bloco, não surgem conflitos entre as constantes de maiúsculas e minúsculas no comutador interno e as constantes no comutador externo.

Pontos importantes relacionados às instruções Switch-Cas

- A opção só pode verificar a igualdade. Isso significa que os outros operadores relacionais, como maior que, são inutilizados em um caso.

- As constantes de caso são avaliadas de cima para baixo, e a primeira constante de caso que corresponde à expressão do comutador é o ponto de entrada de execução. Se nenhuma instrução de interrupção for usada, todo o caso após o ponto de entrada será executado.

- Nenhuma constante de dois casos no mesmo comutador pode ter valores idênticos. Obviamente, uma declaração de chave e uma chave externa anexa podem ter constantes de maiúsculas e minúsculas em comum.

- O caso padrão pode estar localizado no final, meio ou parte superior. Geralmente, o padrão aparece no final de todos os casos.

Declarações de quebra

A instrução break incluída em cada seção do caso determina quando parar de executar as instruções em resposta a um caso correspondente. Sem uma declaração de interrupção em uma seção de caso, depois que uma correspondência é feita, as instruções para essa correspondência e todas as declarações mais abaixo na central são executadas até que uma interrupção ou o final da central seja encontrada. Em algumas situações, isso pode ser exatamente o que você deseja fazer. Caso contrário, você deve

incluir instruções de interrupção para garantir que apenas o código correto seja executado. Às vezes, é desejável ter vários casos em quebras entre eles. Por exemplo, aqui o programa imprime o mesmo valor até que alguma condição atinja.

Fonte: Autoral

Atividade Extra

Vídeo: Operadores em Java

Link: https://www.youtube.com/watch?v=HvwlJT8j7HQ

Referências Bibliográficas

BARNES, D. J. KOLLING, M. **Programação orientada a objetos com java: uma introdução prática usando o bluej.** 4.ed. Pearson: 2009.

FELIX, R. (Org.). **Programação orientada a objetos.** Pearson: 2017.

MEDEIROS, L. F. de. **Banco de dados: princípios e prática.** Intersaberes: 2013;

ORACLE. Java Documentation, 2021. **Documentação oficial da plataforma Java.** Disponível em: < https://docs.oracle.com/en/java/

Ir para exercício