```java
// Figura 5.1: WhileCounter.java
// Repetição controlada por contador com a instrução de repetição while.

public class WhileCounter
{
   public static void main(String[] args)
   {
      int counter = 1; // declara e inicializa a variável de controle

      while (counter <= 10) // condição de continuação do loop
      {
         System.out.printf("%d  ", counter);
         ++counter; // variável de controle de incremento
      }

      System.out.println();
   }
} // fim da classe WhileCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

5.3 Instrução de repetição for

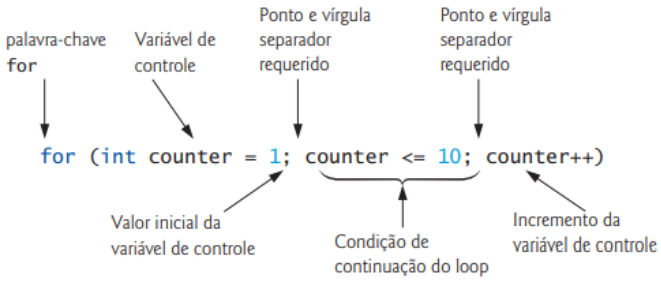
```java
// Figura 5.2: ForCounter.java
// Repetição controlada por contador com a instrução de repetição for.

public class ForCounter
{
   public static void main(String[] args)
   {
      // o cabeçalho da instrução for inclui inicialização,
      // condição de continuação do loop e incremento
      for (int counter = 1; counter <= 10; counter++)
         System.out.printf("%d  ", counter);

      System.out.println();
   }
} // fim da classe ForCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

Uma análise mais atenta do cabeçalho da instrução for



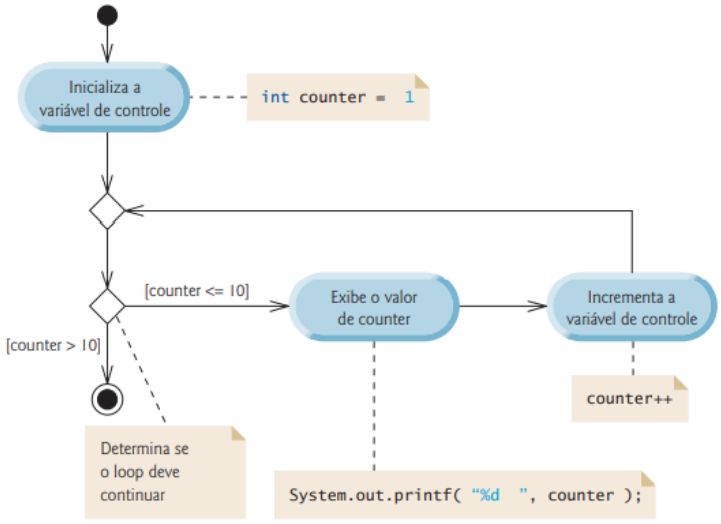
Formato geral de uma instrução for

```
for (inicialização; condiçãoDeContinuaçãoDoLoop; incremento)
   instrução
```

Representando uma instrução for com uma instrução while equivalente

```
inicialização;
while (condiçãoDeContinuaçãoDoLoop)
{
   instrução
   incremento;
}
```

Diagrama de atividades UML para a instrução for



5.4 Exemplos com a estrutura for

a) Varie a variável de controle de 1 a 100 em incrementos de 1.

```
for (int i = 1; i <= 100; i++)
```

b) Varie a variável de controle de 100 a 1 em *decrementos* de 1.

```
for (int i = 100; i >= 1; i--)
```

c) Varie a variável de controle de 7 a 77 em incrementos de 7.

```
for (int i = 7; i <= 77; i += 7)
```

d) Varie a variável de controle de 20 a 2 em *decrementos* de 2.

```
for (int i = 20; i >= 2; i -= 2)
```

e) Varie a variável de controle em relação aos valores 2, 5, 8, 11, 14, 17, 20.

```
for (int i = 2; i <= 20; i += 3)
```

f) Varie a variável de controle em relação aos valores 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.

```
for (int i = 99; i >= 0; i -= 11)
```

```
1  // Figura 5.5: Sum.java
2  // Somando inteiros com a instrução for.
3
4  public class Sum
5  {
6     public static void main(String[] args)
7     {
8        int total = 0;
9
10       // total de inteiros pares de 2 a 20
11       for (int number = 2; number <= 20; number += 2)
12          total += number;
13
14       System.out.printf("Sum is %d%n", total);
15    }
16 } // fim da classe Sum
```

```
Sum is 110
```

Aplicativo: cálculos de juros compostos

Uma pessoa investe US$ 1.000 em uma conta-poupança que rende juros de 5% ao ano. Supondo que todo o juro seja aplicado, calcule e imprima a quantia de dinheiro na conta no fim de cada ano por 10 anos. Utilize a seguinte fórmula para determinar as quantidades:

$$a = p\,(1 + r)^n$$

onde

p é a quantia original investida (isto é, o principal)
r é a taxa de juros anual (por exemplo, utilize 0,05 para 5%)
n é o número de anos
a é a quantia em depósito no fim do n-ésimo ano.

```
1  // Figura 5.6: Interest.java
2  // Cálculos de juros compostos com for.
3
4  public class Interest
5  {
6     public static void main(String[] args)
7     {
8        double amount; // quantia em depósito ao fim de cada ano
9        double principal = 1000.0; // quantidade inicial antes dos juros
10       double rate = 0.05; // taxa de juros
11
12       // exibe cabeçalhos
13       System.out.printf("%s%20s %n", "Year", "Amount on deposit");
14
15       // calcula quantidade de depósito para cada um dos dez anos
16       for (int year = 1; year <= 10; ++year)
17       {
18          // calcula nova quantidade durante ano especificado
19          amount = principal * Math.pow(1.0 + rate, year);
20
21          // exibe o ano e a quantidade
22          System.out.printf("%4d%,20.2f%n", year, amount);
23       }
24    }
25 } // fim da classe Interest
```

```
Year   Amount on deposit
   1          1,050.00
   2          1,102.50
   3          1,157.63
   4          1,215.51
   5          1,276.28
   6          1,340.10
   7          1,407.10
   8          1,477.46
   9          1,551.33
  10          1,628.89
```

```
1  // Figura 5.7: DoWhileTest.java
2  // instrução de repetição do...while.
3
4  public class DoWhileTest
5  {
6     public static void main(String[] args)
7     {
8        int counter = 1;
9
10       do
11       {
12          System.out.printf("%d  ", counter);
13          ++counter;
14       } while (counter <= 10); // fim da instrução do...while
15
16       System.out.println();
17    }
18 } // fim da classe DoWhileTest
```

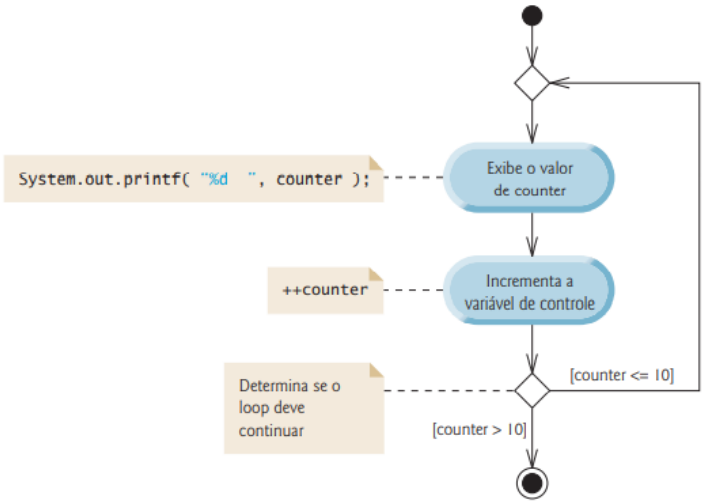
```
1 2 3 4 5 6 7 8 9 10
```

5.5 Instrução de repetição do...while

```
1  // Figura 5.7: DoWhileTest.java
2  // instrução de repetição do...while.
3
4  public class DoWhileTest
5  {
6     public static void main(String[] args)
7     {
8        int counter = 1;
9
10       do
11       {
12          System.out.printf("%d  ", counter);
13          ++counter;
14       } while (counter <= 10); // fim da instrução do...while
15
16       System.out.println();
17    }
18 } // fim da classe DoWhileTest
```

```
1 2 3 4 5 6 7 8 9 10
```

Diagrama de atividades UML para a instrução de repetição do...while



5.6 A estrutura de seleção múltipla switch

```
1  // Figura 5.9: LetterGrades.java
2  // A classe LetterGrades utiliza a instrução switch para contar as letras das notas escolares.
3  import java.util.Scanner;
4
5  public class LetterGrades
6  {
7     public static void main(String[] args)
8     {
9        int total = 0; // soma das notas
10       int gradeCounter = 0; // número de notas inseridas
11       int aCount = 0; // contagem de notas A
12       int bCount = 0; // contagem de notas B
13       int cCount = 0; // contagem de notas C
14       int dCount = 0; // contagem de notas D
15       int fCount = 0; // contagem de notas F
16
17       Scanner input = new Scanner(System.in);
18
```

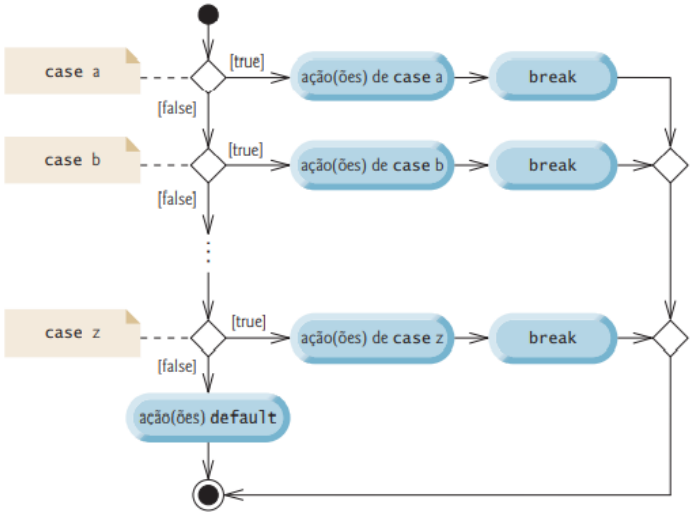
```
19          System.out.printf("%s%n%s%n   %s%n   %s%n",
20             "Enter the integer grades in the range 0-100.",
21             "Type the end-of-file indicator to terminate input:",
22             "On UNIX/Linux/Mac OS X type <Ctrl> d then press Enter",
23             "On Windows type <Ctrl> z then press Enter");
24
25          // faz loop até o usuário inserir o indicador de fim do arquivo
26          while (input.hasNext())
27          {
28             int grade = input.nextInt(); // lê a nota
29             total += grade; // adiciona nota a total
30             ++gradeCounter; // incrementa o número de notas
31
32             // incrementa o contador de letras de nota adequado
33             switch (grade / 10)
34             {
35                case 9:  // a nota estava entre 90
36                case 10: // e 100, inclusivo
37                   ++aCount;
38                   break; // sai do switch
39
40                case 8: // nota estava entre 80 e 89
41                   ++bCount;
42                   break; // sai do switch
43
44                case 7: // nota estava entre 70 e 79
45                   ++cCount;
46                   break; // sai do switch
47
48                case 6: // nota estava entre 60 e 69
49                   ++dCount;
50                   break; // sai do switch
51
52                default: // a nota era menor que 60
53                   ++fCount;
54                   break; // opcional; fecha switch de qualquer maneira
55             } // fim do switch
56          } // fim do while
57
58          // exibe o relatório da nota
59          System.out.printf("%nGrade Report:%n");
60
61          // se usuário inseriu pelo menos uma nota...
62          if (gradeCounter != 0)
63          {
64             // calcula a média de todas as notas inseridas
65             double average = (double) total / gradeCounter;
66
67             // gera a saída de resumo de resultados
68             System.out.printf("Total of the %d grades entered is %d%n",
69                gradeCounter, total);
70             System.out.printf("Class average is %.2f%n", average);
71             System.out.printf("%n%s%n%s%d%n%s%d%n%s%d%n%s%d%n%s%d%n",
72                "Number of students who received each grade:",
73                "A: ", aCount,   // exibe número de notas A
74                "B: ", bCount,   // exibe número de notas B
75                "C: ", cCount,   // exibe número de notas C
76                "D: ", dCount,   // exibe número de notas D
77                "F: ", fCount); // exibe número de notas F
78          } // fim do if
79          else // nenhuma nota foi inserida, assim gera a saída da mensagem apropriada
80             System.out.println("No grades were entered");
81       } // fim de main
82    } // finaliza a classe letterGrades
```

```
Enter the integer grades in the range 0-100.
Type the end-of-file indicator to terminate input:
   On UNIX/Linux/Mac OS X type <Ctrl> d then press Enter
   On Windows type <Ctrl> z then press Enter
99
92
45
57
63
71
76
85
90
100
^Z

Grade Report:
Total of the 10 grades entered is 778
Class average is 77.80
Number of students who received each grade:

A: 4
B: 1
C: 2
D: 1
F: 2
```

Diagrama de atividades UML para a instrução `switch`



5.7 Estudo de caso da classe `AutoPolicy`: `Strings` em instruções `switch`

Você foi contratado por uma companhia de seguros de automóvel que atende estes estados do nordeste dos Estados Unidos — Connecticut, Maine, Massachusetts, New Hampshire, Nova Jersey, Nova York, Pensilvânia, Rhode Island e Vermont. A empresa quer que você crie um programa que produz um relatório indicando para cada uma das apólices de seguro de automóvel se a apólice é válida em um estado com seguro de automóvel "sem culpa" (modalidade de seguro em que o segurado é indenizado independentemente de sua responsabilidade no sinistro) — Massachusetts, Nova Jersey, Nova York e Pensilvânia.

Classe `AutoPolicy`

A classe `AutoPolicy` (Figura 5.11) representa uma apólice de seguro de automóvel. A classe contém:

- A variável de instância `int accountNumber` (linha 5) para armazenar o número da conta da apólice.
- A variável de instância `String` de `makeAndModel` (linha 6) para armazenar a marca e o modelo do carro (como um `"Toyota Camry"`).
- A variável de instância `String` de `state` (linha 7) para armazenar a sigla do estado de dois caracteres que representa o estado em que a apólice é válida (por exemplo, `"MA"` significando Massachusetts).
- Um construtor (linhas 10 a 15) que inicializa as variáveis de instância da classe.

- Os métodos setAccountNumber e getAccountNumber (linhas 18 a 27) para *definir* e *obter* uma variável de instância `accountNumber` de `AutoPolicy`.
- Os métodos setMakeAndModel e getMakeAndModel (linhas 30 a 39) para *definir* e *obter* a variável de instância `AutoPolicy` de um `makeAndModel`.
- Os métodos `setState` e `getState` (linhas 42 a 51) para *definir* e *obter* a variável de instância `AutoPolicy` de um `state`.
- O método `isNoFaultState` (linhas 54 a 70) para retornar um valor `boolean` que indica se a apólice é válida em um estado de seguros de automóvel "sem culpa"; observe o nome do método — a convenção de nomeação para um método *get* que retorna um valor `boolean` é começar o nome com `"is"` em vez de `"get"` (esse método é comumente chamado de *método de predicado*).

No método `isNoFaultState`, a expressão de controle da instrução `switch` (linha 59) é a `String` retornada por método `getState` de `AutoPolicy`. A instrução `switch` compara o valor da expressão de controle com os rótulos `case` (linha 61) para determinar se a apólice é válida em Massachusetts, Nova Jersey, Nova York ou Pensilvânia (os estados "sem culpa"). Se houver uma correspondência, então a linha 62 configura a variável local `noFaultState` como `true` e a instrução `switch` termina; caso contrário, o caso `default` define `noFaultState` como `false` (linha 65). Então, o método `isNoFaultState` retorna o valor da variável local `noFaultState`.

Para simplificar, não validamos os dados de `AutoPolicy` no construtor ou nos métodos *set*, e supomos que as abreviaturas dos estados sempre têm duas letras maiúsculas. Além disso, uma classe `AutoPolicy` real provavelmente conteria muitas outras variáveis de instância e métodos para dados como o nome, endereço do titular da conta etc. No Exercício 5.30, você será solicitado a aprimorar a classe `AutoPolicy` validando a abreviação do estado utilizando as técnicas que você aprenderá na Seção 5.9.

```
1  // Figura 5.11: AutoPolicy.java
2  // Classe que representa uma apólice de seguro de automóvel.
3  public class AutoPolicy
4  {
5     private int accountNumber; // número da conta da apólice
6     private String makeAndModel; // carro ao qual a apólice é aplicada
7     private String state; // abreviatura do estado com duas letras
8
9     // construtor
10    public AutoPolicy(int accountNumber, String makeAndModel, String state)
11    {
12       this.accountNumber = accountNumber;
13       this.makeAndModel = makeAndModel;
14       this.state = state;
15    }
16
17    // define o accountNumber
18    public void setAccountNumber(int accountNumber)
19    {
20       this.accountNumber = accountNumber;
21    }
22
23    // retorna o accountNumber
24    public int getAccountNumber()
25    {
26       return accountNumber;
27    }
28
29    // configura o makeAndModel
30    public void setMakeAndModel(String makeAndModel)
31    {
32       this.makeAndModel = makeAndModel;
33    }
34
35    // retorna o makeAndModel
36    public String getMakeAndModel()
37    {
38       return makeAndModel;
39    }
40
41    // define o estado
42    public void setState(String state)
43    {
44       this.state = state;
45    }
46
47    // retorna o estado
48    public String getState()
49    {
50       return state;
51    }
52
```

```
53      // método predicado é retornado se o estado tem seguros "sem culpa"
54      public boolean isNoFaultState()
55      {
56         boolean noFaultState;
57
58         // determina se o estado tem seguros de automóvel "sem culpa"
59         switch (getState()) // obtém a abreviatura do estado do objeto AutoPolicy
60         {
61            case "MA": case "NJ": case "NY": case "PA":
62               noFaultState = true;
63               break;
64            default:
65               noFaultState = false;
66               break;
67         }
68
69         return noFaultState;
70      }
71   } // fim da classe AutoPolicy
```

Classe AutoPolicyTest

```
1   // Figura 5.12: AutoPolicyTest.java
2   // Demonstrando Strings em um switch.
3   public class AutoPolicyTest
4   {
5      public static void main(String[] args)
6      {
7         // cria dois objetos AutoPolicy
8         AutoPolicy policy1 =
9            new AutoPolicy(11111111, "Toyota Camry", "NJ");
10        AutoPolicy policy2 =
11           new AutoPolicy(22222222, "Ford Fusion", "ME");
12
13        // exibe se cada apólice está em um estado "sem culpa"
14        policyInNoFaultState(policy1);
15        policyInNoFaultState(policy2);
16     }
17
18     // método que mostra se um AutoPolicy
19     // está em um estado com seguro de automóvel "sem culpa"
20     public static void policyInNoFaultState(AutoPolicy policy)
21     {
22        System.out.println("The auto policy:");
23        System.out.printf(
24           "Account #: %d; Car: %s; State %s %s a no-fault state%n%n",
25           policy.getAccountNumber(), policy.getMakeAndModel(),
26           policy.getState(),
27           (policy.isNoFaultState() ? "is": "is not"));
28     }
29  } // fim da classe AutoPolicyTest
```

```
The auto policy:
Account #: 11111111; Car: Toyota Camry;
State NJ is a no-fault state

The auto policy:
Account #: 22222222; Car: Ford Fusion;
State ME is not a no-fault state
```

5.8 Instruções break e continue

```
1  // Figura 5.13: BreakTest.java
2  // a instrução break sai de uma instrução for.
3  public class BreakTest
4  {
5     public static void main(String[] args)
6     {
7        int count; // variável de controle também utilizada depois que loop termina
8
9        for (count = 1; count <= 10; count++) // faz o loop 10 vezes
10       {
11          if (count == 5)
12             break; // termina o loop se a contagem for 5
13
14          System.out.printf("%d ", count);
15       }
16
17       System.out.printf("%nBroke out of loop at count = %d%n", count);
18    }
19 } // fim da classe BreakTest
```

```
1 2 3 4
Broke out of loop at count = 5
```

```
1  // Figura 5.14: ContinueTest.java
2  // Instrução continue que termina uma iteração de uma instrução for.
3  public class ContinueTest
4  {
5     public static void main(String[] args)
6     {
7        for (int count = 1; count <= 10; count++) // faz o loop 10 vezes
8        {
9           if (count == 5)
10             continue; // pula o código restante no corpo do loop se a contagem for 5
11
12          System.out.printf("%d ", count);
13       }
14
15       System.out.printf("%nUsed continue to skip printing 5%n");
16    }
17 } // fim da classe ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

5.9 Operadores lógicos

Operador E condicional (&&)

```
if ((gender == FEMALE) && (age >= 65))
   ++seniorFemales;
```

expressão1	expressão2	expressão1 && expressão2
false	false	false
false	true	false
true	false	false
true	true	true

Operador OU condicional (||)

```
if ((semesterAverage >= 90) || (finalExam >= 90))
   System.out.println ("Student grade is A");
```

expressão1	expressão2	expressão1 || expressão2
false	false	false
false	true	true
true	false	true
true	true	true

OU exclusivo lógico booleano (^)

expressão1	expressão2	expressão1 ^ expressão2
false	false	false
false	true	true
true	false	true
true	true	false

Operador de negação lógica (!)

```
if (! (grade == sentinelValue))
   System.out.printf("The next grade is %d%n", grade);
```

```
if (grade != sentinelValue)
   System.out.printf("The next grade is %d%n", grade);
```

expressão	!expressão
false	true
true	false

```
1  // Figura 5.19: LogicalOperators.java
2  // Operadores lógicos.
3
4  public class LogicalOperators
5  {
6     public static void main(String[] args)
7     {
8        // cria a tabela-verdade para o operador && (E condicional)
9        System.out.printf("%s%n%s: %b%n%s: %b%n%s: %b%n%s: %b%n%n",
10          "Conditional AND (&&)", "false && false", (false && false),
11          "false && true", (false && true),
12          "true && false", (true && false),
13          "true && true", (true && true));
14
15       // cria a tabela-verdade para o operador || (OU condicional)
16       System.out.printf("%s%n%s: %b%n%s: %b%n%s: %b%n%s: %b%n%n",
17          "Conditional OR (||)", "false || false", (false || false),
18          "false || true", (false || true),
19          "true || false", (true || false),
20          "true || true", (true || true));
21
22       // cria a tabela-verdade para o operador & (E lógico booleano)
23       System.out.printf("%s%n%s: %b%n%s: %b%n%s: %b%n%s: %b%n%n",
24          "Boolean logical AND (&)", "false & false", (false & false),
25          "false & true", (false & true),
26          "true & false", (true & false),
27          "true & true", (true & true));
28
29       // cria a tabela-verdade para o operador | (OU inclusivo lógico booleano)
30       System.out.printf("%s%n%s: %b%n%s: %b%n%s: %b%n%s: %b%n%n",
31          "Boolean logical inclusive OR (|)",
32          "false | false", (false | false),
33          "false | true", (false | true),
34          "true | false", (true | false),
35          "true | true", (true | true));
36
```

```
37          // cria a tabela-verdade para o operador ^ (OU exclusivo lógico booleano)
38          System.out.printf("%s%n%s: %b%n%s: %b%n%s: %b%n%s: %b%n%n",
39             "Boolean logical exclusive OR (^)",
40             "false ^ false", (false ^ false),
41             "false ^ true", (false ^ true),
42             "true ^ false", (true ^ false),
43             "true ^ true", (true ^ true));
44
45          // cria a tabela-verdade para o operador ! (negação lógica)
46          System.out.printf("%s%n%s: %b%n%s: %b%n", "Logical NOT (!)",
47             "!false", (!false), "!true", (!true));
48       }
49    } // fim da classe LogicalOperators
```

```
Conditional AND (&&)
false && false: false
false && true: false
true && false: false
true && true: true

Conditional OR (||)
false || false: false
false || true: true
true || false: true
true || true: true

Boolean logical AND (&)
false & false: false
false & true: false
true & false: false
true & true: true

Boolean logical inclusive OR (|)
false | false: false
false | true: true
true | false: true
true | true: true

Boolean logical exclusive OR (^)
false ^ false: false
false ^ true: true
true ^ false: true
true ^ true: false

Logical NOT (!)
!false: true
!true: false
```

Os operadores de precedência e associatividade apresentados até agora

Operadores	Associatividade	Tipo
`++ --`	da direita para a esquerda	unário pós-fixo
`++ -- + - ! (tipo)`	da direita para a esquerda	unário pré-fixo
`* / %`	da esquerda para a direita	multiplicativo
`+ -`	da esquerda para a direita	aditivo
`< <= > >=`	da esquerda para a direita	relacional
`== !=`	da esquerda para a direita	igualdade
`&`	da esquerda para a direita	E lógico booleano
`^`	da esquerda para a direita	OU exclusivo lógico booleano
`|`	da esquerda para a direita	OU inclusivo lógico booleano
`&&`	da esquerda para a direita	E condicional
`||`	da esquerda para a direita	OU condicional
`?:`	da direita para a esquerda	ternário condicional
`= += -= *= /= %=`	da direita para a esquerda	atribuição