

Estrutura de repetição de pré-teste “while”.

A estrutura de repetição “**while**” vai “prender” o fluxo do programa em um determinado bloco de código, fazendo com que esse seja executado repetidas vezes essa estrutura é chamada de loop de repetição pré-teste pois testa uma condição antes de entrar em “**loop**”. O loop “**while**” é executado enquanto a condição for verdadeira “**true**”, logo se o primeiro teste resultar em falso “**false**” o loop não executa nenhuma vez e o fluxo do programa segue ignorando o código no “**corpo**” do “**loop**” “**while**”.

Sintaxe:

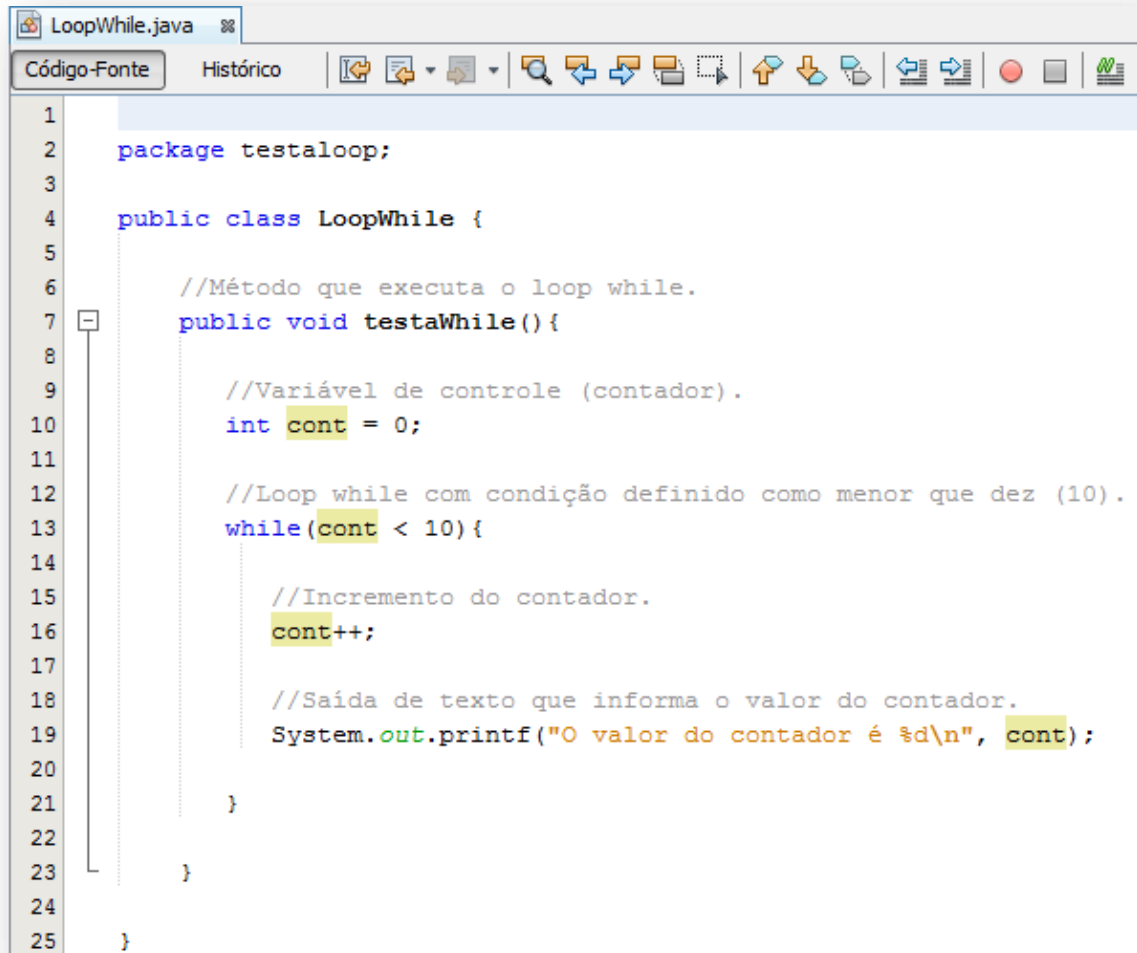
```
while(teste_lógico){  
    bloco_de_código_caso_verdadeiro  
}
```

Então podemos dizer que o loop “**while**” vai executar enquanto sua condição (teste lógico) resultar em “**true**”, logo se essa condição nunca mudar o “**loop**” será “**eterno**”. Devemos então definir uma forma do “**loop**” parar, essa forma é chamada de contadores, esses contadores nada mais são do que variáveis que mudam ou podem mudar de valor a cada interação do “**loop**”, essas variáveis são testadas na condição do “**loop**” a cada nova interação desse. Existem dois conceitos de variáveis de contador no “**loop**” “**while**”, são elas:

Loop controlado por variável de contador: Geralmente é uma variável do tipo inteiro que começa em zero (0), mas esse valor pode ser diferente depende muito da necessidade que levou a criação do “**loop**”. Na condição do “**loop**” (teste lógico) essa variável é testada, geralmente com operadores relacionais de comparação (<, <=, >, >=, == ou !=) e no caso do “**loop**” “**while**” enquanto a condição for verdadeira uma nova interação do “**loop**” será executada. A cada vez que o loop é executado o valor da variável de contador é incrementado ou decrementado dependendo novamente da necessidade pela qual o “**loop**” foi criado o que define sua condição para executar ou não uma ou mais interações.

A imagem abaixo ilustra um exemplo simples de um loop “**while**” controlado por contador que conta até dez (10):

Classe “LoopWhile.java”:



```
1
2 package testaloop;
3
4 public class LoopWhile {
5
6     //Método que executa o loop while.
7     public void testaWhile(){
8
9         //Variável de controle (contador).
10        int cont = 0;
11
12        //Loop while com condição definido como menor que dez (10).
13        while(cont < 10){
14
15            //Incremento do contador.
16            cont++;
17
18            //Saída de texto que informa o valor do contador.
19            System.out.printf("O valor do contador é %d\n", cont);
20
21        }
22    }
23 }
24
25 }
```

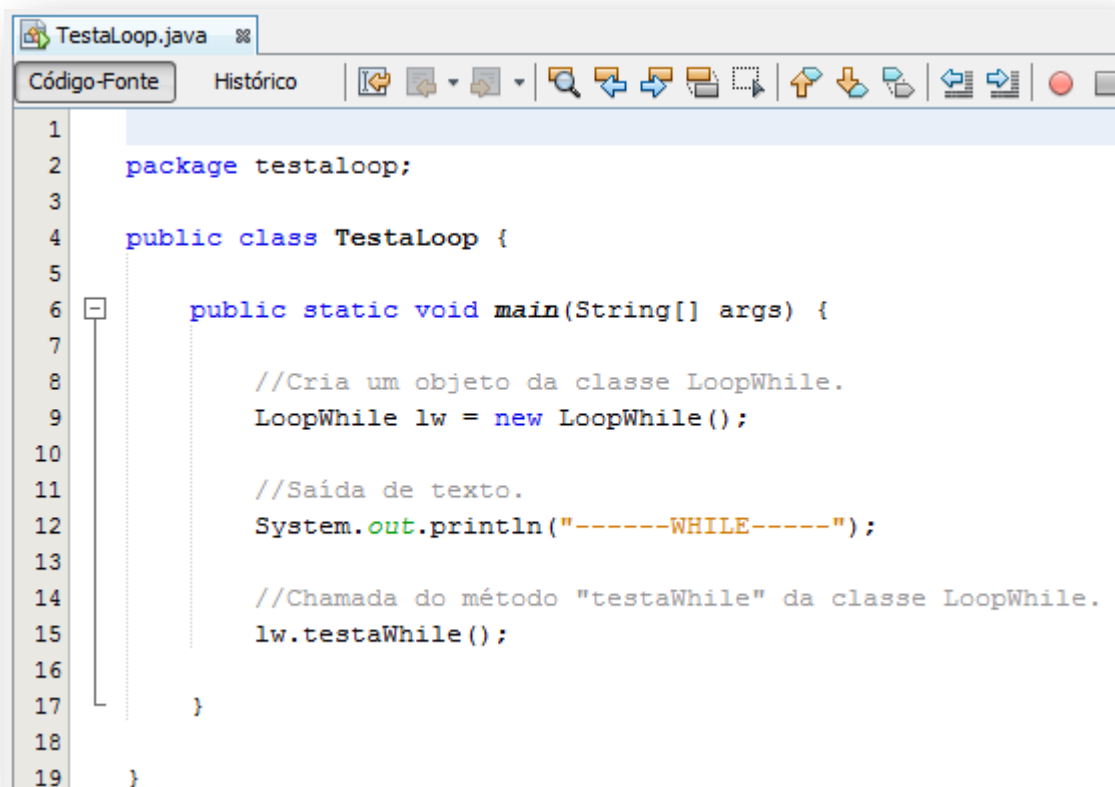
A linha dez (10) declara uma variável local dentro do método “testaWhile” e inicializa essa com zero (0).

A linha treze (13) define o cabeçalho do “loop” configurando sua condição de execução em “enquanto a variável **cont** for menor que dez (10)”, sempre que essa condição (teste lógico) retornar “**true**” (verdadeiro) o “**loop**” executa uma interação.

A linha dezesseis (16) incrementa a variável de controle por contador em um a cada interação do “**loop**” é esse processo que garante que em algum momento (quando o valor da variável “**cont**” alcançar dez (10) o loop vai cessar.

A linha dezenove (19) apenas exibem um texto informando o valor da variável de controle “**cont**”.

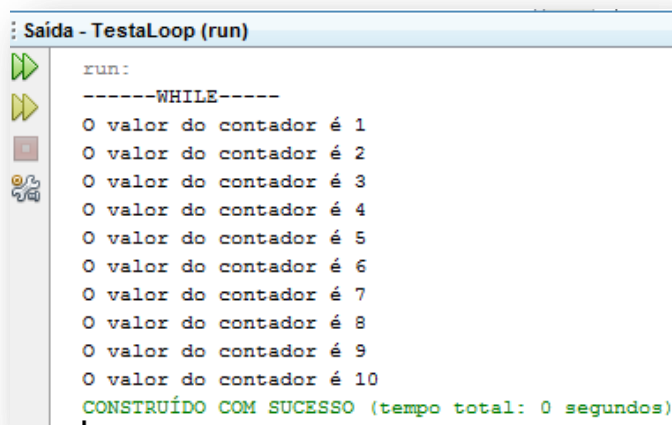
Criação de um objeto da classe “LoopWhile.java”:



```
1
2 package testaloop;
3
4 public class TestaLoop {
5
6     public static void main(String[] args) {
7
8         //Cria um objeto da classe LoopWhile.
9         LoopWhile lw = new LoopWhile();
10
11         //Saída de texto.
12         System.out.println("-----WHILE-----");
13
14         //Chamada do método "testaWhile" da classe LoopWhile.
15         lw.testaWhile();
16
17     }
18
19 }
```

Aqui temos apenas a criação e execução do objeto e do método da classe “LoopWhile.java”.

Resultado da execução da classe “LoopWhile.java”:



```
Saída - TestaLoop (run)
run:
-----WHILE-----
O valor do contador é 1
O valor do contador é 2
O valor do contador é 3
O valor do contador é 4
O valor do contador é 5
O valor do contador é 6
O valor do contador é 7
O valor do contador é 8
O valor do contador é 9
O valor do contador é 10
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Loop controlado por variável de sentinela: Nessa variação do “loop” não sabemos quantas vezes o “loop” vai ser executado então ao invés de usar uma variável que “conta” o número de vezes que o “loop” vai ser executado usamos uma variável que “espera” que um determinado valor seja informado ou pelo usuário ou por outra fonte qualquer.

A imagem abaixo ilustra a aplicação do exemplo simples de um loop “**while**” controlado por sentinela que só encerra quando o usuário digitar “s”:

```
1
2 package testaloop;
3
4 import java.util.Scanner;
5
6 public class LoopWhile2 {
7
8     //Variável de sentinela.
9     String status;
10
11     public void testeWhile2(){
12
13         //Cria um objeto da classe Scanner para receber dados digitados pelo
14         //usuário.
15         Scanner sc = new Scanner(System.in);
16
17         //Solicita uma entrada de dados.
18         System.out.println("Digite s para sair.");
19
20         //Recebe a entrada de dados.
21         status = sc.next();
22
23         //Loop while com condição baseada em variável de sentinela.
24         while(!(status.equalsIgnoreCase("s"))){
25
26             //Solicita uma entrada de dados.
27             System.out.println("Digite s para sair.");
28
29             //Recebe a entrada de dados.
30             status = sc.next();
31
32         }
33     }
34 }
```

A linha quinze (15) declara um objeto da classe “**Scanner**” necessário para as entradas de dados.

As linhas dezoito (18) e vinte (20) solicitam e recebem respectivamente um valor para o usuário que nesse caso define se o “**loop**” vai ou não ser executado.

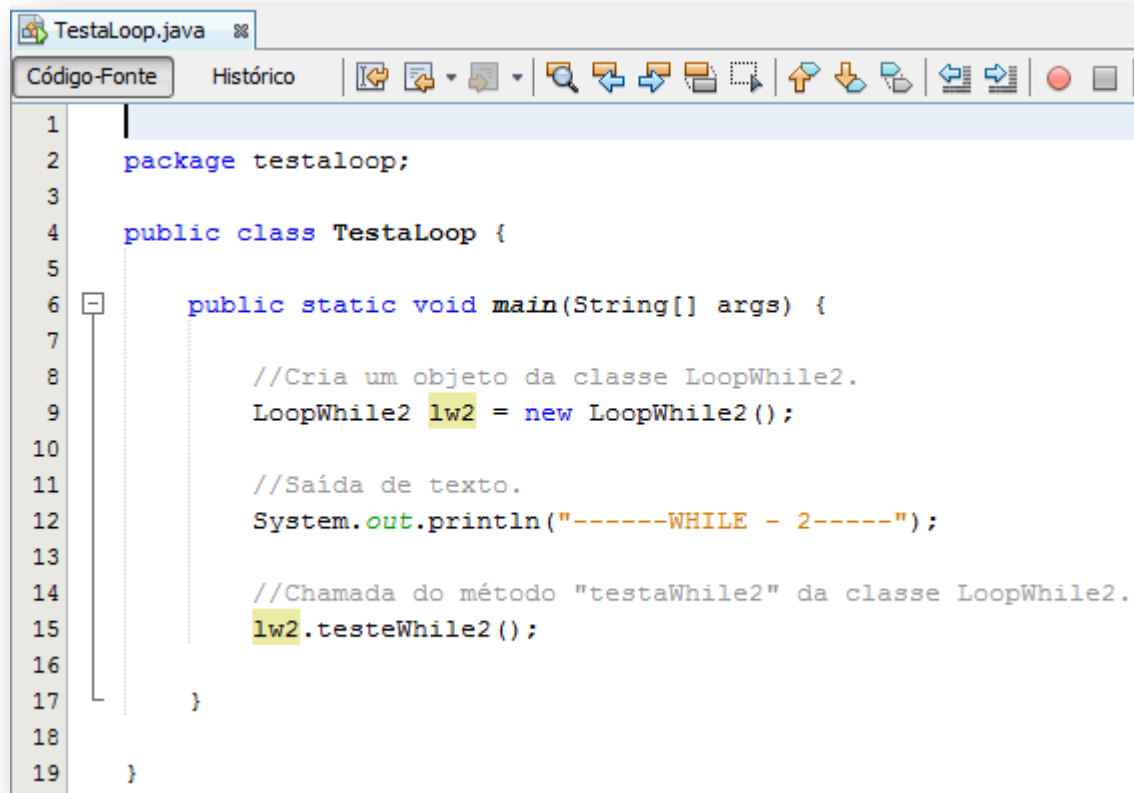
A linha vinte e quatro (24) implementa um loop de repetição “**while**” controlado por variável de sentinela, observe que no JAVA quando precisamos comparar uma “**string**” não usamos os operadores relacionais tradicionais, esse não são capazes de comparar “**strings**”, isso ocorre porque no JAVA o tipo “**string**” não é um tipo de primitivo, mas uma classe essa classe a classe “**String**” por sua vez possui vários métodos responsáveis por fazer tarefas de comparação. No

caso desse loop usamos o método “**equalsIgnoreCase**” que desconsidera questões de letras maiúsculas e minúsculas no processo de comparação, passamos então como parâmetro a **string** “**s**” a ser comparada com os valores digitados pelo usuário, sempre que “**s**” for capturado seja maiúsculo ou minúsculo o método “**equalsIgnoreCase**” “**true**” (verdadeiro) e caso contrário retornara “**false**” (falso), porem queremos que essa lógica seja invertida daí a utilização de um operador de negação lógica “**!**”, logo quando o usuário digitar qualquer letra diferente de “**s**” o resultado será falso, porem o operador lógico vai inverter esse resultado para verdadeiro o que causa uma nova interação do loop e quando o usuário digitar “**s**” o método retornará verdadeiro, porem nesse caso o operador de negação vai inverter o resultado para falso encerrando assim a execução do “**loop**”.

As linhas vinte e sete (27) e trinta (30) solicitam e recebem respectivamente um valor para o usuário que nesse caso define se o “**loop**” vai ou não ser executado, poderíamos inclusive ter apenas essas linhas solicitando valores para o usuário e remover as que estão fora do “**loop**”.

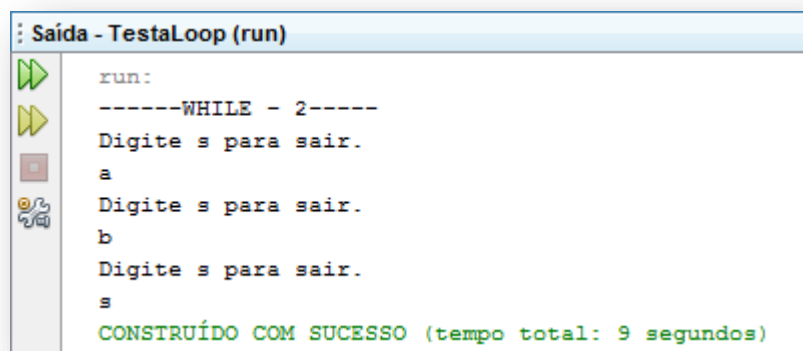
Observação: A variável de sentinela poderia esperar também por um número ao invés de uma “**String**”, veremos isso na aplicação do conceito no programa do caixa eletrônico.

Criação de um objeto da classe “LoopWhile.java”:



```
1  
2 package testaloop;  
3  
4 public class TestaLoop {  
5  
6     public static void main(String[] args) {  
7  
8         //Cria um objeto da classe LoopWhile2.  
9         LoopWhile2 lw2 = new LoopWhile2();  
10  
11        //Saída de texto.  
12        System.out.println("-----WHILE - 2-----");  
13  
14        //Chamada do método "testaWhile2" da classe LoopWhile2.  
15        lw2.testeWhile2();  
16    }  
17  
18 }  
19 }
```

Resultado da execução da classe “LoopWhile2.java”:



```
run:  
-----WHILE - 2-----  
Digite s para sair.  
a  
Digite s para sair.  
b  
Digite s para sair.  
s  
CONSTRUÍDO COM SUCESSO (tempo total: 9 segundos)
```

Estrutura de controle de repetição pós-teste “do...while”.

A estrutura de repetição de pós-teste “do...while” funciona de forma muito semelhante a sua antecessora desse curso a única diferença é que nessa estrutura a condição (teste lógico) para a continuação ou parada do loop é feita no final e não no início como no loop “while” isso garante que o loop vai

executar pelo menos uma vez independentemente do valor da variável de controle.

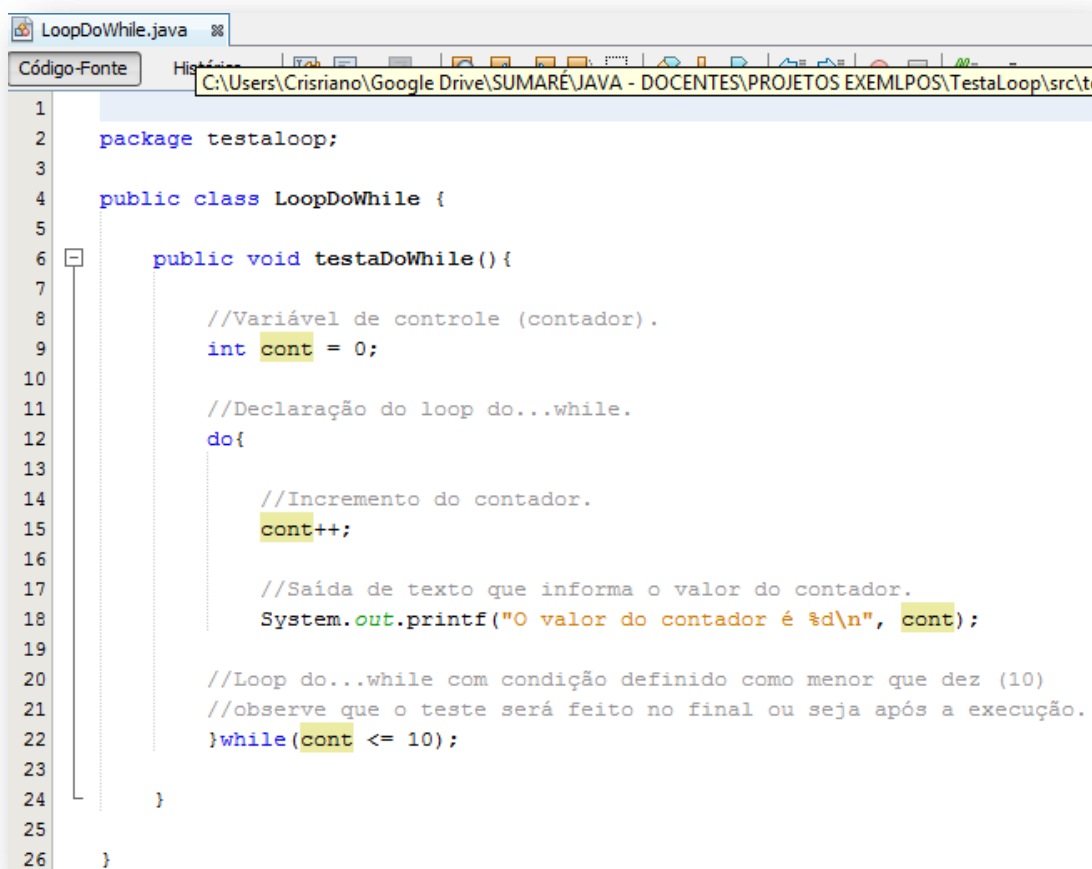
O “**loop**” “**do...while**” implementa os mesmos conceitos de variáveis de controle por contador e por sentinela assim como no “**while**”.

do{

bloco_de_código_caso_verdadeiro

} while(teste_lógico);

Classe “LoopWhile.java”:



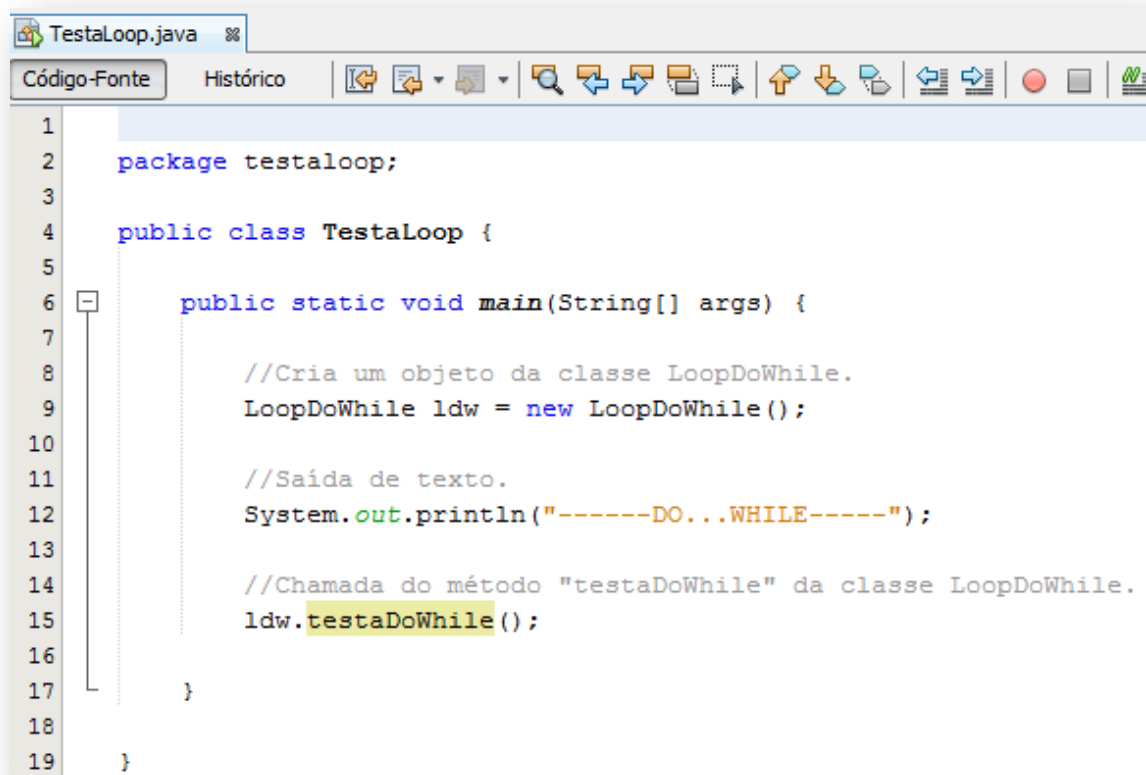
```
1 package testaloop;
2
3
4 public class LoopDoWhile {
5
6     public void testaDoWhile(){
7
8         //Variável de controle (contador).
9         int cont = 0;
10
11         //Declaração do loop do...while.
12         do{
13
14             //Incremento do contador.
15             cont++;
16
17             //Saída de texto que informa o valor do contador.
18             System.out.printf("O valor do contador é %d\n", cont);
19
20             //Loop do...while com condição definido como menor que dez (10)
21             //observe que o teste será feito no final ou seja após a execução.
22         }while(cont <= 10);
23
24     }
25
26 }
```

Observe que existem muitas semelhanças entre o “**do...while**” e o “**while**” a linha nove (9) declara e inicia um contador e a linha quinze (15) incrementa esse contador até aqui nenhuma novidade.

A linha doze (12) declara o início do loop com o uso da palavra reservada “**do**” seguida da abertura do corpo do “**loop**” com as chaves “{” e essa é fechada na linha vinte e dois “}”, observe que é depois da abertura e fechamento do corpo que encontramos o comando “**while**” seguido da condição que define se o “**loop**” deve ou não continuar essa forma de estruturar é que define a principal

característica do “**loop**” de executar primeiro e testar depois garantindo assim que pelo menos uma vez vai haver interação com o “**loop**”.

Criação de um objeto da classe “LoopDoWhile.java”:



```
1
2 package testaloop;
3
4 public class TestaLoop {
5
6     public static void main(String[] args) {
7
8         //Cria um objeto da classe LoopDoWhile.
9         LoopDoWhile ldw = new LoopDoWhile();
10
11         //Saída de texto.
12         System.out.println("-----DO...WHILE-----");
13
14         //Chamada do método "testaDoWhile" da classe LoopDoWhile.
15         ldw.testaDoWhile();
16     }
17 }
18
19 }
```

O processo de criação do objeto e chamada do método que executa o “**loop**” “**do...while**” é idêntico ao exemplo anterior.

Resultado da execução da classe “LoopDoWhile.java”:


```
Saída - TestaLoop (run)
run:
-----DO...WHILE-----
O valor do contador é 1
O valor do contador é 2
O valor do contador é 3
O valor do contador é 4
O valor do contador é 5
O valor do contador é 6
O valor do contador é 7
O valor do contador é 8
O valor do contador é 9
O valor do contador é 10
O valor do contador é 11
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

A saída também não muda nada dos exemplos anteriores isso porque o valor inicial do contador é zero (0), porém se esse valor for trocado para dez (10) tanto no exemplo do “**while**” quanto no exemplo do “**do...while**” veremos que o primeiro não vai executar nenhuma vez enquanto o segundo vai executar uma vez.

Observação: Poderíamos implementar o mesmo conceito de variável de controle por sentinela do loop “**while**” no loop “**do...while**”, mas dada semelhança entre ambos não vou documentar isso aqui.

Estrutura de controle de repetição “for”.

Essa estrutura de repetição é capaz de realizar as mesmas tarefas que “**while**” e “**do...while**” a diferença aqui é a forma como a variável de controle é tratada nas questões de declaração e incremento/decremento assim como a condição do “loop” essas tarefas são todas declaradas no cabeçalho (momento da declaração) do “**loop**”.

Sintaxe:

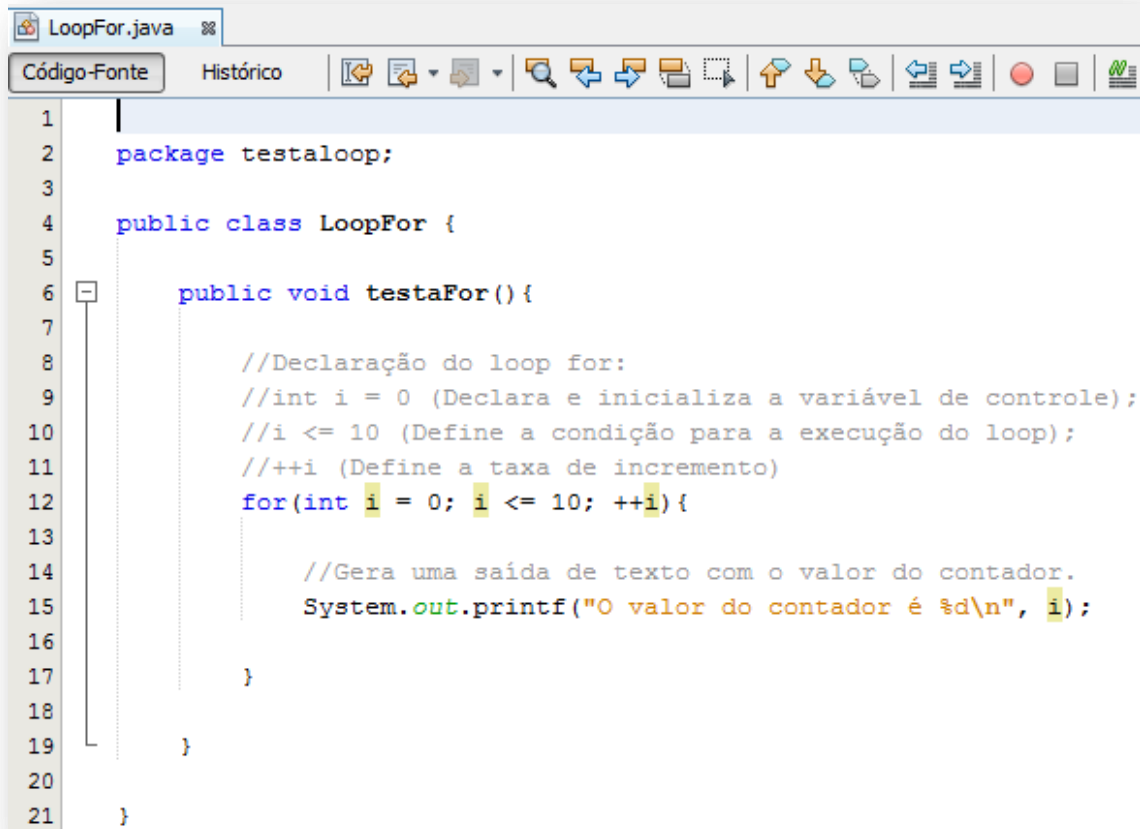
```
for(declaração_do_contador; teste_lógico; incremento/decremento){
    bloco_de_código_caso_verdadeiro
}
```

Observe que após “**for**(“ iniciamos a declaração e inicialização da variável de controle, por exemplo “**int i = 0**” em seguida definimos a condição para execução ou parada do “**loop**”, por exemplo “**i <= 10**” e finalmente a taxa (passo) de incremento/decremento, por exemplo “**i++**” assim como nos outros “loops” a abertura e o fechamento do corpo do “**loop**” é feito através das chaves “{” abertura e “}” fechamento e tudo que estiver entre essa será executado a cada passagem do “**loop**”.

Observação: a declaração e inicialização da variável é feita apenas na primeira vez que o “**loop**” for executado.

Observação: o processo de incremento/decremento é ignorado na primeira execução do “loop”.

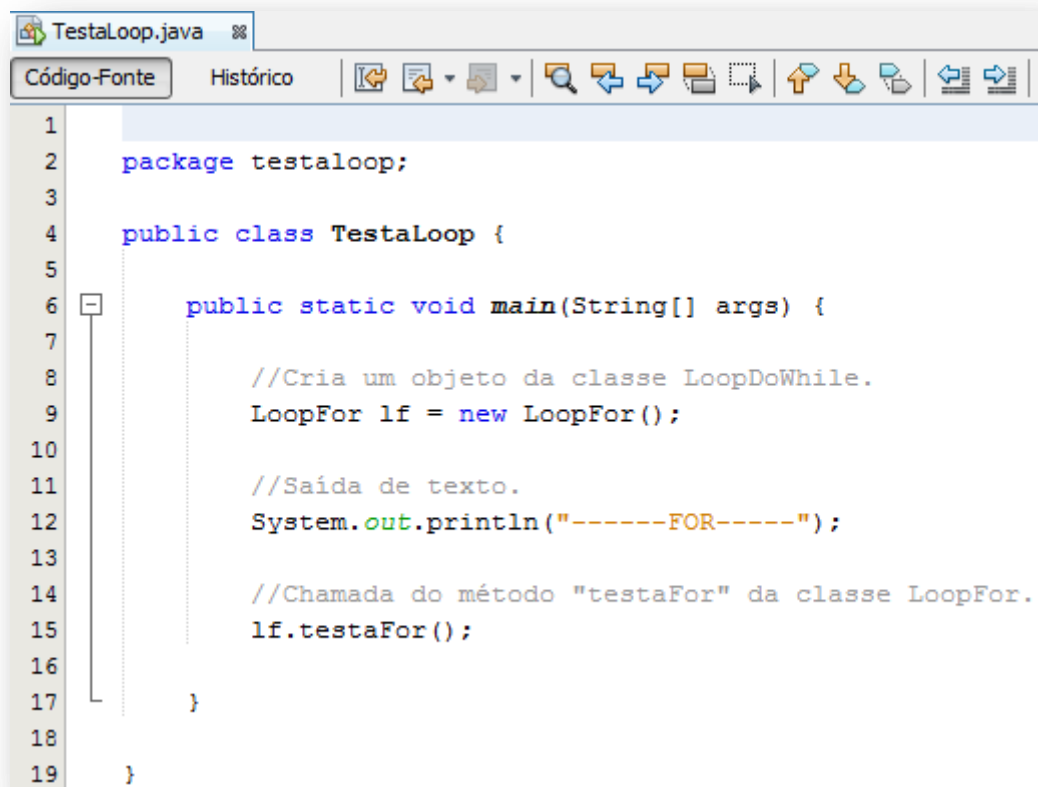
Classe “LoopFor.java”:



```
1  
2 package testalooop;  
3  
4 public class LoopFor {  
5  
6     public void testaFor() {  
7  
8         //Declaração do loop for:  
9         //int i = 0 (Declara e inicializa a variável de controle);  
10        //i <= 10 (Define a condição para a execução do loop);  
11        //++i (Define a taxa de incremento)  
12        for(int i = 0; i <= 10; ++i) {  
13  
14            //Gera uma saída de texto com o valor do contador.  
15            System.out.printf("O valor do contador é %d\n", i);  
16  
17        }  
18  
19    }  
20  
21 }
```

Observe na linha doze (12) a declaração do “**loop for**” seguindo a ordem, declaração e inicialização de contado, definição da condição de continuidade ou parada do loop e nesse caso um incremento, note o uso do ponto e virgula “;” usado na separação de cada etapa.

Criação de um objeto da classe “LoopFor.java”:



```
1
2 package testaloop;
3
4 public class TestaLoop {
5
6     public static void main(String[] args) {
7
8         //Cria um objeto da classe LoopDoWhile.
9         LoopFor lf = new LoopFor();
10
11         //Saída de texto.
12         System.out.println("-----FOR-----");
13
14         //Chamada do método "testaFor" da classe LoopFor.
15         lf.testaFor();
16
17     }
18
19 }
```

O processo de criação do objeto e chamada do método que executa o “**loop**” “**for**”.

Resultado da execução da classe “LoopFor.java”:

```
Saída - TestaLoop (run)
run:
-----FOR-----
O valor do contador é 0
O valor do contador é 1
O valor do contador é 2
O valor do contador é 3
O valor do contador é 4
O valor do contador é 5
O valor do contador é 6
O valor do contador é 7
O valor do contador é 8
O valor do contador é 9
O valor do contador é 10
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Aqui temos a saída da execução do “**loop for**”, observe que a contagem começa em zero (0) indicando que a primeira execução do “loop” não houve incremento e observe também que o “**for**” contou até dez (10), ou seja o contador foi de zero (0) até dez (10) resultando assim e onze (11) “**passagens**” se quiséssemos apenas dez (10) sendo de zero (0) a nove (9) teríamos que mudar o valor da variável de controle para um (1). Essa característica faz com que o esse loop seja muito usado para “**correr**” os índices de vetores que na maioria das linguagens de programação começa seus índices sempre em zero (0).

Observação: O “**for**” também trabalha com o conceito de variável de controle por sentinela, porem isso é pouco usual e não vou abordar aqui dado o fato de que os “**loops**” “**while**” e “**do...while**” são mais aptos para tal tarefa.

Aplicando o conceito de “loop” no programa de caixa eletrônico.

Em nossa última “atualização” do programa do caixa eletrônico nós adicionamos um “menu” onde o usuário digitava uma opção para realizar uma ação sendo que “1 – Saldo”, “2 – Depósito” e “3 – Saque”.

A questão é o usuário só pode realizar uma ação por execução do programa se houver necessidade de realizar duas ou mais ações o programa deve ser executado novamente, vamos então usar um “loop” de repetição controlado por sentinela que vai dar uma quarta opção “4 – Sair” e enquanto o usuário não digitar sair vai ser possível executar qualquer uma das outras três opções quantas vezes forem necessárias ou possíveis.

A imagem abaixo ilustra o código com as alterações proposta:

```
1 package caixaeletronicosumare;
2
3 import utilitarios.Conta;
4 import java.util.Scanner;
5
6 public class CaixaEletronicoSumare {
7
8     public static void main(String[] args) {
9
10         Conta objConta = new Conta();
11         Scanner sc = new Scanner(System.in);
12
13         int opc = 0;
14         double valor;
15
16         while (opc != 4) {
17
18             System.out.println("Digite uma opção:");
19             System.out.println("1 - Saldo");
20             System.out.println("2 - Deposito");
21             System.out.println("3 - Saque");
22             System.out.println("4 - Sair");
23
24             opc = sc.nextInt();
25
26             switch (opc) {
27
28                 case 1:
29                     objConta.saldo();
30                     break;
31
32                 case 2:
33                     System.out.println("Digite um valor de deposito:");
34                     valor = sc.nextDouble();
35                     objConta.deposito(valor);
36                     objConta.saldo();
37                     break;
38
39                 case 3:
40                     System.out.println("Digite um valor de saque");
41                     valor = sc.nextDouble();
42                     objConta.sacar(valor);
43                     objConta.saldo();
44                     break;
45
46                 case 4:
47                     System.exit(0);
48
49                 default:
50                     System.out.println("Opção inválida");
51
52             }
53
54         }
55
56     }
57
58 }
```

Observe que nas linhas treze (13) e quatorze (14) estão declaradas respectivamente as variáveis **"opc"** que recebe e armazena a opção de operação digitada pelo usuário e a variável **"valor"** que recebe valores usados nos métodos de saque e depósito, declarar essas variáveis nesse ponto do

código as deixam fora do **“loop”** e da estrutura **“switch”** permitindo assim que ambas possa manipular essas variáveis.

Na linha dezesseis (16) é declarado um **“loop”** de repetição **“while”** que usa como condição uma variável de controle por sentinela **“while (opc != 4)”**, como não sabemos quantas vezes o **“loop”** vai executar, pois isso depende de quantas operações o usuário pretende realizar definimos que quando esse escolher a opção sair **“4 – Sair”** o teste lógico vai retornar **“false”** falso, fazendo com que o **“case 4:”** nas linhas **46-47** seja executado lançando um comando **“System.exit(0)”** que encerra a aplicação.

A estrutura **“switch”** fecha na linha cinquenta e dois (52) e o **“while”** na linha cinquenta e quatro (54).