

Laços

Roland Teodorowitsch

Fundamentos de Programação - Escola Politécnica - PUCRS

4 de maio de 2023

Introdução

Objetivos

- Implementar laços com `while`, `for` e `do`
- Acompanhar a execução de um programa manualmente
- Familiarizar-se com os algoritmos comuns com laços
- Entender laços aninhados
- Implementar programas que leem e processam conjuntos de dados

Conteúdos

- O Laço `while`
- Exercícios
- O Laço `for`
- O Laço `do`
- Sentinelas de Processamento
- Algoritmos Comuns com Laços
- Laços Aninhados
- Resumo

O Laço `while`

Laços

- Um laço serve para repetir a execução de trechos de um programa
- Laços executam instruções repetidamente enquanto uma condição for verdadeira
- Exemplos de aplicações com laços
 - Executar um processamento específico sobre um conjunto predefinido de dados

Leia o raio de 10 círculos e mostre a área e a circunferência de cada círculo.
 - Executar um processamento específico sobre um conjunto de dados cujo tamanho foi fornecido

Leia quantos círculos serão processados, e a seguir leia o raio de cada círculo, calculando a sua área e circunferência.
 - Executar um processamento específico sobre um conjunto indeterminado de dados

Leia o raio de um círculo e mostre a sua área e a sua circunferência, repetindo esses passos enquanto o raio lido for maior do que zero.
 - Executar alguns processamentos e algoritmos específicos

Somatório, produtório, contagem, média, cálculo de juros compostos, etc.
 - Realizar ações repetidas diversas dentro de aplicações

Movimentar um objeto dentro de um jogo, imprimir tabelas com várias linhas, desenhar gráficos, executar simulações, etc.
 - etc.

O Laço while

- O comando `while` pode ser usado em Java para implementar um laço

```
while ( expressaoLogica ) {  
    // comandos  
}
```

- `while` é composto por:
 - Teste de permanência (uma expressão lógica)
 - Comandos que serão executados repetidamente enquanto o teste for `true`
- Assim, para contar de 1 até 5, por exemplo, poderíamos fazer:

```
int i = 1;           // valor inicial atribuído a uma variável  
while ( i <= 5 ) {   // testa se os comandos do laço serão executados ou não  
    System.out.println(i); // comandos que serão repetidos  
    i = i + 1;         // atualização da variável de controle do laço  
}
```

Mais exemplos de contagem...

- Contar de 10 até 50:

```
int i = 10;
while ( i <= 50 ) {
    System.out.println(i);
    i = i + 1;
}
```

- Contar de 5 até 50, de 5 em 5:

```
int i = 5;
while ( i <= 50 ) {
    System.out.println(i);
    i = i + 5;
}
```

- Contar de 10 até 1:

```
int i = 10;
while ( i >= 1 ) {
    System.out.println(i);
    i = i - 1;
}
```

- Contar de -10 até -100, de 2 em 2:

```
int i = -10;
while ( i >= -100 ) {
    System.out.println(i);
    i = i - 2;
}
```


Mais exemplos (1)

- Leia o raio de 10 círculos e mostre a área e a circunferência de cada círculo.

```
import java.util.Scanner;
/** Programa que lê o raio de 10 círculos e mostra a área e a circunferência de cada círculo. */
public class Circulo1 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int i = 0;           // Variável de controle/indução inicializada fora do laço
        while ( i < 10 ) {   // Teste para verificar se os comandos ainda devem ser executados
            System.out.print("Raio do círculo? ");    double raio = in.nextDouble();
            double area = Math.PI * raio * raio;       System.out.printf("Área=%f\n", area);
            double circ = 2.0 * Math.PI * raio;        System.out.printf("Circunferência=%f\n", circ);
            i = i + 1;    // Atualização da variável de controle
        }
    }
}
```

- Se a condição do while nunca se tornar false, o laço será infinito!!!
- Variáveis declaradas dentro do laço são criadas a cada iteração e NÃO existem fora do while
- Recomenda-se atualizar a variável de controle do laço no final dos comandos

Mais exemplos (2)

- Leia quantos círculos serão processados, e a seguir leia o raio de cada círculo, calculando a sua área e circunferência.

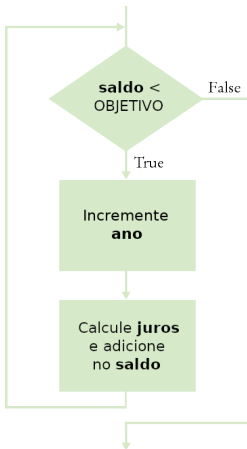
```
import java.util.Scanner;
/**
 * Programa que lê quantos círculos serão processados, e, a seguir,
 * lê o raio de cada círculo, calculando a sua área e circunferência.
 */
public class Circulo2 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Número de círculos? ");
        int numCirculos = in.nextInt();
        int i = 0;
        while ( i < numCirculos ) {
            System.out.print("Raio do círculo? ");
            double raio = in.nextDouble();
            double area = Math.PI * raio * raio;
            System.out.printf("Área=%f\n", area);
            double circ = 2.0 * Math.PI * raio;
            System.out.printf("Circunferência=%f\n", circ);
            i = i + 1;
        }
    }
}
```

Mais exemplos (3)

- Leia o raio de um círculo e mostre a sua área e a sua circunferência, repetindo esses passos enquanto o raio lido for maior do que zero.

```
import java.util.Scanner;
/**
 * Programa que lê o raio de um círculo e mostra a sua área e a sua circunferência,
 * repetindo esses passos enquanto o raio lido for maior do que zero.
 */
public class Circulo3 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int i = 0;
        System.out.print("Raio do círculo? ");
        double raio = in.nextDouble();
        while ( raio > 0.0 ) {
            double area = Math.PI * raio * raio;
            System.out.printf("Área=%f\n", area);
            double circ = 2.0 * Math.PI * raio;
            System.out.printf("Circunferência=%f\n", circ);
            System.out.print("Raio do círculo? ");
            raio = in.nextDouble(); // raio NÃO deve ser redeclarado!
        }
    }
}
```

Planejando um Laço while



- Cálculo de juros compostos (Unidade 1)

Inicie com um valor de ano igual a zero e um total de \$10.000
Repita o seguinte enquanto o total seja menor do que \$20.000
 Adicione 1 ao valor do ano
 Multiplique o total por 1,05 (crescimento de 5%)
Informe o último valor atribuído ao ano como resposta

- Em Java:

```
while (saldo < ALVO) {  
    ano++;  
    double juros = saldo * TAXA_DE_JUROS/100.0;  
    saldo = saldo + juros;  
}
```

DobrandoInvestimento.java

```
/**
 * Calcula o número de anos necessário para dobrar o valor inicial de um investimento.
 * Adaptado de Horstmann (2013, p. 143).
 */
public class DobrandoInvestimento {
    public static void main(String[] args) {
        final double TAXA_DE_JUROS = 5.0;
        final double SALDO_INICIAL = 10000.00;
        final double OBJETIVO = 2.0 * SALDO_INICIAL;
        double saldo = SALDO_INICIAL;
        int ano = 0;
        // Conta o número de anos necessários para dobrar o valor inicialmente investido
        while (saldo < OBJETIVO) {
            ano++;
            double juros = saldo * TAXA_DE_JUROS / 100.0;
            saldo = saldo + juros;
        }
        System.out.println("O valor investido dobra após " + ano + " anos.");
    }
}
```

Resultado:

O valor investido dobra após 15 anos.

Exemplos de Laços com while (1)

Laço	Saída	Explicação
<pre>i = 0; soma = 0; while (soma < 10) { i++; soma = soma + i; System.out.println(i+" "+soma); }</pre>	<pre>1 1 2 3 3 6 4 10</pre>	Quando <code>soma</code> for igual a 10, a condição do laço será falsa, e o laço se encerra.
<pre>i = 0; soma = 0; while (soma < 10) { i++; soma = soma - i; System.out.println(i+" "+soma); }</pre>	<pre>1 -1 2 -3 3 -6 4 -10 ...</pre>	Como <code>soma</code> nunca atinge 10, isto consiste em um “laço infinito”.
<pre>i = 0; soma = 0; while (soma < 0) { i++; soma = soma - i; System.out.println(i+" "+soma); }</pre>	(Nenhuma saída)	A condição <code>soma < 0</code> é falsa quando é testada pela primeira vez, e o laço nunca é executado.

Exemplos de Laços com `while` (2)

Laço	Saída	Explicação
<pre>i = 0; soma = 0; while (soma >= 10) { i++; soma = soma + i; System.out.println(i+" "+soma); }</pre>	(Nenhuma saída)	O programador provavelmente pensou: "Pare quando a soma for pelo menos 10". Entretanto a condição do laço controla quando o laço é executado, e não quando ele se encerra.
<pre>i = 0; soma = 0; while (soma < 10) ; { i++; soma = soma + i; System.out.println(i+" "+soma); }</pre>	(Nenhuma saída e o programa nunca termina)	Observe que o ponto-e-vírgula após o teste do laço faz com que o corpo do laço corresponda a um comando vazio. O programa executará para sempre pois <code>soma < 0</code> e este valor não será atualizado no corpo do laço (comando vazio).

Erros Comuns: NÃO pense “Já chegamos lá?”

- O corpo do laço somente será executado se a condição de teste for verdadeira
- Então a lógica correta é “Continuo executando o laço?”
- Se `saldo` deve crescer até que alcance `OBJETIVO`, qual versão executará o corpo do laço corretamente?

```
while (saldo < OBJETIVO) {  
    ano++;  
    juros = saldo * TAXA;  
    saldo = saldo + juros;  
}
```

```
while (saldo >= OBJETIVO) {  
    ano++;  
    juros = saldo * TAXA;  
    saldo = saldo + juros;  
}
```


Erros Comuns: laços infinitos

- O corpo do laço será executado até que a condição de teste se torne falsa
- O que acontece se você se esquecer de atualizar a variável de teste?
 - `saldo` é a variável de teste (`OBJETIVO` é constante)
 - Seu programa ficará no laço para sempre! (ou até que você pare o programa)

```
while (saldo < OBJETIVO) {  
    ano      ++;  
    juros = saldo * TAXA;  
    // saldo = saldo + juros;  
}
```

Erros Comuns: erros de limite

- Uma variável do tipo contadora é frequentemente usada na condição de teste
- A variável contadora pode iniciar em 0 ou 1 (programadores frequentemente iniciam contadores com 0)
- Se você quer contar 5 dedos, qual código deve ser usado?

```
// Inicia em 0, usa-se <
int dedo = 0;
final int DEDOS = 5;
while (dedo < DEDOS) {
    System.out.println(dedo);
    dedo++;
}
// 0,1,2,3,4
```

```
// Inicia em 1, usa-se <=
int dedo = 1;
final int DEDOS = 5;
while (dedo <= DEDOS) {
    System.out.println(dedo);
    dedo++;
}
// 1,2,3,4,5
```

Resumo do Laço `while`

- Laços com `while` são usados com grande frequência
- Inicialize as variáveis antes do teste
- A condição é testada ANTES do corpo do laço
 - Isto é chamado pré-teste
 - A condição frequentemente usa uma variável contadora
- Algo dentro do corpo do laço deve alterar uma das variáveis usadas no teste
- Cuidado com laços infinitos!

Exercícios

Exercícios (1)

- 1 Faça laços em Java usando `while` para mostrar:
- a Os números inteiros de 1 a 10, inclusive
 - b Os números inteiros de 100 a 200, inclusive.
 - c Os números inteiros de 10 a 1, inclusive, em ordem regressiva.
 - d Os números inteiros de -10 a -50, inclusive.
 - e Os números pares entre a e b (com $a \geq b$).
 - f As 10 primeiras potências de 2.

Exercícios (2)

- 2 O código a seguir calcula a soma dos dígitos de um número (por exemplo, para 1729 o valor seria $1+7+2+9$). Acompanhe a execução desse código, instrução por instrução, e mostre como os valores das variáveis `n`, `sum` e `digit` se alteram ao longo da execução.

```
int n = 1729;
int sum = 0;
while (n > 0) {
    int digit = n % 10;
    sum = sum + digit;
    n = n / 10;
}
System.out.println(sum);
```

Exercícios (3)

- 3 Escreva laços `while` em Java, declarando todas as variáveis utilizadas, para:
- a Ler 20 pares de valores (a e b) escrevendo qual é o maior valor.
 - b Calcular a soma dos valores de 1 até 20.
 - c Mostrar os elementos de uma progressão aritmética de n elementos que inicia em a e tem razão r .
 - d Calcular a soma dos elementos do item anterior.
 - e Mostrar os elementos de uma progressão geométrica de n elementos que inicia em a e tem razão r .
 - f Calcular a soma dos elementos do item anterior.
 - g Calcular o fatorial de um número inteiro lido do terminal.
 - h Ler um número inteiro e escrever se ele é primo ou não.

Exercícios (4)

- 4 Faça o teste de mesa para o trecho de programa em Java a seguir, mostrando todas as alterações de valores nas variáveis declaradas e todas as saídas de terminal, e considerando que os valores lidos do teclado serão respectivamente 2, 4, 3, 2, 0, 2.

```
Scanner in = new Scanner(System.in);
int x, a, n, z;

x = in.nextInt();
n = in.nextInt();
while (x > 0) {
    a = 1;
    while (a <= n) {
        z=a*n;
        System.out.println(z);
        a = a + 1;
    }
    x = in.nextInt();
    n = in.nextInt();
}
```


O Laço for

O Laço for

- Em Java, tudo que é feito com `while` pode ser feito também com `for`
- Mas use laços `for` quando
 - Houver uma variável de indução com início, atualização e fim claramente identificáveis
 - For interessante deixar o código mais conciso (controle do laço em uma única linha)
- Por exemplo, para fazer o somatório dos números de 1 até 10 poderíamos fazer:

```
int soma = 0;
// versao com while
int i = 1; // inicializacao
while (i <= 10) { // teste
    soma = soma + i;
    i++; // atualizacao
}
```

```
int soma = 0;
// versao com for
for (int i = 1; i <= 10; i++) {
    soma = soma + i;
}
```

Sintaxe do Comando `for`

```
for ( inicializacao; condicao; atualizacao)  
    corpo;
```

- O comando `for` tem 4 partes:
 - 1 Inicialização: é executada uma vez antes do laço iniciar
 - 2 Condição de permanência: é verificada antes de cada iteração (deve ser verdadeira)
 - 3 Corpo do laço (bloco de comandos ou comando único): executado enquanto a condição `for` verdadeira
 - 4 Atualização: executada sempre depois do bloco de comandos e antes de se fazer um novo teste da condição
- Depois da inicialização, o laço `for` repetirá um ciclo formado por
 - Teste da condição — Execução do corpo do laço — Atualização

Execução de um laço for

1 Initialize counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter);  
}
```

2 Check condition

counter = 1

```
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter);  
}
```

3 Execute loop body

counter = 1

```
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter);  
}
```

4 Update counter

counter = 2

```
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter);  
}
```

5 Check condition again

counter = 2

```
for (counter = 1; counter <= 10; counter++)  
{  
    System.out.println(counter);  
}
```

Exemplos de Laços for (1)

- Contar de 1 (inclusive) até 5 (inclusive) [5 iterações: 1 2 3 4 5]

```
for (int i = 1; i<=5; i++) System.out.println(i);  
for (int i = 1; i<6; i++) System.out.println(i);
```

- Contar de 1 (inclusive) até 5 (exclusive) [4 iterações: 1 2 3 4]

```
for (int i = 1; i<=4; i++) System.out.println(i);  
for (int i = 1; i<5; i++) System.out.println(i);
```

- Contar de 0 (inclusive) até 5 (inclusive) [6 iterações: 0 1 2 3 4 5]

```
for (int i = 0; i<=5; i++) System.out.println(i);  
for (int i = 0; i<6; i++) System.out.println(i);
```

- Contar de 0 (inclusive) até 5 (exclusive) [5 iterações: 0 1 2 3 4]

```
for (int i = 0; i<=4; i++) System.out.println(i);  
for (int i = 0; i<5; i++) System.out.println(i);
```

Exemplos de Laços for (2)

- Contagem regressiva [6 iterações: 5 4 3 2 1 0]

```
for (int i = 5; i>=0; i--) System.out.println(i);
```

- Incremento igual a 2 [5 iterações: 0 2 4 6 8]

```
for (int i = 0; i<9; i=i+2) System.out.println(i);
```

- Razão geométrica igual a 2 [5 iterações: 1 2 4 8 16]

```
for (int i = 1; i<=20; i=i*2) System.out.println(i);
```

- Percorrer todas as letras de uma cadeia de caracteres [4 iterações: J A V A]

```
String s = "JAVA";  
for (int i = 0; i<s.length(); i++) System.out.println(s.charAt(i));
```

Planejando um Laço for



- Considerando um valor inicial investido de 10000, e uma taxa de juros anual de 5%, escreva um programa que: leia o número total de anos de investimento (`numAnos`) e, para cada ano transcorrido, imprima o número do ano e o saldo total ao final desse ano.
- Por exemplo, para `numAnos` igual a 5, o programa deve imprimir:
 - ano 1: 10500.00
 - ano 2: 11025.00
 - ano 3: 11576.25
 - ano 4: 12155.06
 - ano 5: 12762.82

```
for (int ano = 1; ano <= numAnos; ano++) {  
    // Atualiza saldo  
    // Imprime ano e saldo  
}
```

Investimento.java

```
import java.util.Scanner;

/**
 * Este programa imprime uma tabela mostrando o crescimento anual de um investimento.
 */
public class Investimento {
    public static void main(String[] args) {
        final double TAXA = 5.0;
        final double SALDO_INICIAL = 10000;
        double saldo = SALDO_INICIAL;

        System.out.print("Quantos anos? ");
        Scanner in = new Scanner(System.in);
        int numAnos = in.nextInt();

        for (int ano = 1; ano <= numAnos; ano++) {
            double juros = saldo * TAXA / 100.0;
            saldo = saldo + juros;
            System.out.printf("ano %d: %.2f\n", ano, saldo);
        }
    }
}
```


Cuidados a serem tomados

- Limite final do laço: inclusive ou exclusive?
- Variável de indução: crescente ou decrescente?
 - Com valores crescentes usa-se \leq ou $<$
 - Com valores decrescentes usa-se \geq ou $>$
- Condições erradas podem gerar **laços infinitos**:

```
// 0 2 4 6 8 10 ...  
for (int i=0; i!=9; i=i+2) System.out.println(i);  
  
// 0 -1 -2 -3 -4 -5 ...  
for (int i=0; i<10; --i) System.out.println(i);
```

- Evite atualizar o contador no corpo do laço for:

```
for (int i=1; i<=100; i++) {  
    if (i % 10 == 0)           // Pular valores divisíveis por 10  
        i++;                  // Estilo ruim para for...  
    System.out.println(i);  
}
```

Escopo de Variáveis do Laço `for`

- Escopo é o “tempo de vida” de uma variável.
- Quando a variável `x` é declarada no comando `for`, ela existe apenas dentro do bloco do laço

```
for ( int x = 1; x < 10; x = x + 1) {  
    // comandos a serem executados dentro do laço  
    // 'x' pode ser usado em qualquer lugar dentro deste bloco  
}  
if (x > 100)    // Erro! 'x' esta fora de escopo!
```

- Solução: declarar `'x'` fora do laço

```
int x;  
for ( x = 1; x < 10; x = x + 1) {}
```

Resumo do Laço `for`

- Laços com `for` são muito usados
- Eles têm uma notação bastante concisa
 - Inicialização ; Condição ; Atualização ; Corpo do laço
 - A inicialização acontece uma única vez no início do laço
 - A condição é testada todas as vezes ANTES (pré-teste) de executar o corpo do laço
 - O incremento é realizado APÓS o corpo do laço
- Use laços `for` seguindo o modelo padrão
- Adequado para contagens inteiras, processamento de *strings*, vetores ou matrizes, etc.

Exercícios sobre Laço `for`

Exercícios 1 e 2

- 1 Escreva laços `for` em Java, declarando todas as variáveis utilizadas, para:
 - a. Mostrar os valores de 1 até 10.
 - b. Mostrar os valores de 10 até 1, em ordem regressiva.
 - c. Calcular a soma dos valores de 1 até 20.
 - d. Calcular o fatorial de um número inteiro lido do terminal.
 - e. Ler 20 pares de valores (*a* e *b*) escrevendo qual é o maior valor.
 - f. Ler um número inteiro e escrever se ele é primo ou não.
- 2 Escreva um programa em Java para ler o número de alunos de uma turma e a seguir ler as notas destes alunos na prova da disciplina, determinando e imprimindo: a média da turma, a nota mais baixa e a nota mais alta.

Exercício 3

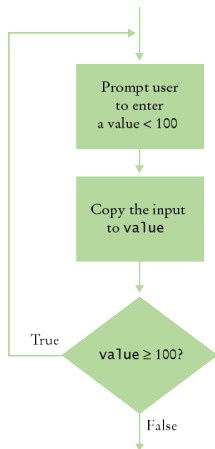
- 3 Faça o teste de mesa para o trecho de programa em Java a seguir, mostrando todas as alterações de valores nas variáveis declaradas e todas as saídas de terminal, e considerando que os valores lidos do teclado serão respectivamente 4, 2, 2, 3, 2, 0.

```
Scanner in = new Scanner(System.in);
int x, a, n, z;

n = in.nextInt();
for ( x = in.nextInt() ; x > 0 ; x = in.nextInt() ) {
    for ( a = 1 ; a <= n ; a = a + 1 ) {
        z=a*n;
        System.out.println(z);
    }
    n = in.nextInt();
}
```

O Laço do

O Laço do



- Usa-se o laço `do` quando se deseja executar o corpo do laço pelo menos uma vez, testando a condição APÓS a primeira repetição do laço

```
int i = 1; // inicializacao
final int DEDOS = 5;
do {
    // comandos...
    i++; // atualizacao
} while (i <= DEDOS); // teste
```


Exemplo de Laço do

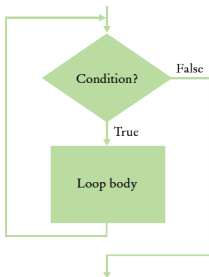
- Validação da entrada de usuários
 - Verificar se um valor lido está dentro do limite
 - O usuário tem que fornecer alguma entrada antes para ser validada

```
int valor;  
do {  
    System.out.println("Forneca um valor inteiro < 100: ");  
    valor = in.nextInt();  
} while (valor >= 100); // Teste
```

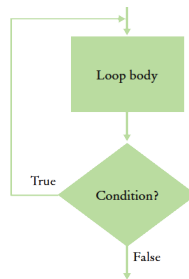
Dica de Programação

- Fluxogramas para laços: evite código "spaghetti"(nunca faça uma seta apontar para dentro do corpo de um laço)

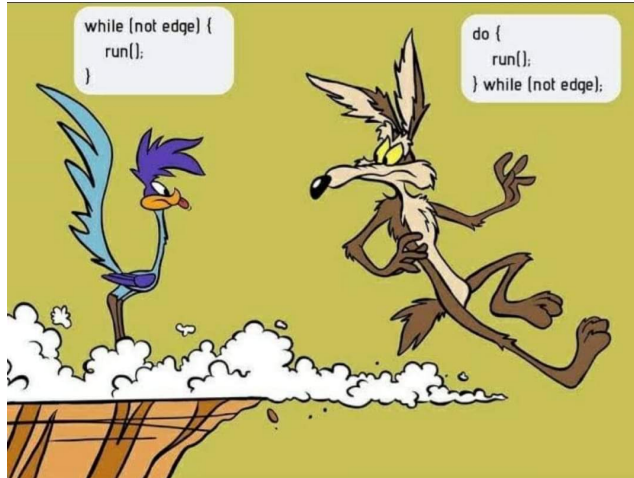
`while` e `for` testam antes



`do` testa depois



Humor



Sentinelas de Processamento

Sentinelas de Processamento

- Valores de sentinela indicam o final de um conjunto de dados, mas não fazem parte dos dados
- Podem ser usados em muitos casos
 - Quando não se sabe quantos itens há em uma lista, usa-se um caracter ou valor especial para sinalizar que não há mais itens
 - Para entradas de números positivos, é comum usar o valor -1:

```
int numFuncionarios = 0;
double salario = in.nextDouble();
double soma = 0.0;
while (salario != -1) {
    soma = soma + salario;
    numFuncionarios++;
    salario = in.nextDouble();
}
```

Calculando a Média de um Conjunto Indeterminado de Valores

- Declare e inicialize uma variável `soma` com 0
- Declare e inicialize uma variável `contagem` com 0
- Declare e inicialize uma variável `entrada` com 0
- Mostre uma mensagem solicitando que o usuário forneça os dados
- Fique no laço até que o valor de sentinela seja fornecido
 - Leia um valor e salve em `entrada`
 - Se `entrada` não for igual a -1
 - Adicione `entrada` em `sum`
 - Adicione 1 na variável `contagem`
- Tenha certeza de que pelo menos um valor for fornecido antes de fazer a divisão
 - Divida `sum` por `contagem` e mostre o resultado
- Fim

Sentinelas.java

```
import java.util.Scanner;

/**
 * Este programa imprime a média de um conjunto de salários terminados com um valor de sentinela.
 * Adaptado de Horstmann (2013, p. 158-159).
 */
public class Sentinela {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        double soma = 0.0, salario = 0.0;
        int numSalarios = 0;
        System.out.print("Forneça um conjunto de salários (use -1 para terminar): ");
        while (salario != -1) { // Processa os dados até encontrar a sentinela
            salario = in.nextDouble();
            if (salario != -1) {
                soma = soma + salario;
                numSalarios++;
            }
        }
        if (numSalarios > 0) { // Cálculo e impressão
            double media = soma / numSalarios;
            System.out.printf("Salario médio: R$%.2f\n", media);
        }
        else { System.out.println("Sem dados..."); }
        in.close();
    }
}
```

Variáveis Booleanas e Sentinelas

- Uma variável booleana (frequentemente chamada de *flag* pode ser usada para controlar um laço)

```
System.out.print("Forneça um conjunto de valores (use -1 para te  
boolean concluido = false;  
while (!concluido) {  
    double valor = in.nextDouble();  
    if (valor == -1) {  
        concluido = true;  
    }  
    else {  
        // Processa o valor  
    }  
}
```


Para permitir qualquer valor numérico...

- Se os valores válidos puderem ser negativos ou positivos, não se pode usar -1 (ou qualquer outro número) como sentinela
- A solução então é usar qualquer outra sentinela não numérica
- Como `in.nextDouble` falha para valores não numéricos, deve-se usar `in.hasNextDouble` antes
 - Retorna um booleano: `true`, se a entrada estiver correta (for um número), ou `false`, se a entrada não for um número
 - Em caso de `true`, pode-se usar `in.nextDouble`

```
System.out.print("Forneça valores reais (digite Q para encerrar)");
while (in.hasNextDouble()) {
    double valor = in.nextDouble();
    // Processa o valor...
}
```

Algoritmos Comuns com Laços

Algoritmos Comuns com Laços

- Somatório
- Produtório
- Valor médio
- Contagem de ocorrências
- Encontrar a primeira ocorrência
- Perguntar até que uma ocorrência seja encontrada
- Máximo e mínimo
- Comparar valores adjacentes

Somatório

- Inicialize `total` com 0
- Pode-se usar o laço com sentinela
- Acrescente o valor em `total`

```
double total = 0;
while (in.hasNextDouble()) {
    double entrada = in.nextDouble();
    total = total + entrada;
}
```

Produtório

- Inicialize `produto` com 1
- Multiplique o valor por `produto`

```
double produto = 1;
while (in.hasNextDouble()) {
    double entrada = in.nextDouble();
    produto = produto * entrada;
}
```

Valor Médio

- Faça o somatório dos valores
- Inicialize `contagem` com 0, incrementando-a a cada valor lido
- Verifique o valor de `contagem` antes da divisão!

```
double total = 0;
int contagem = 0;
while (in.hasNextDouble()) {
    double entrada = in.nextDouble();
    total = total + entrada;
    contagem++;
}
if (contagem > 0) {
    double media = total / contagem;
    System.out.println("Media = " + media);
}
```

Contagem de Ocorrências

- Inicialize contagem com 0
- Use um laço for
- Incremente o contador a cada ocorrência

```
int letrasMaiusculas = 0;
for (int i = 0; i < str.length(); i++) {
    char ch = str.charAt(i);
    if (Character.isUpperCase(ch)) {
        letrasMaiusculas++;
    }
}
```

Encontrar a Primeira Ocorrência

- Inicialize uma variável sentinela/booleana com `false`
- Inicialize o contador de posições com 0 (primeiro caracter do *string*)
- Use uma condição composta no laço
- Laços com pré-teste tratam a situação para *string* vazio

```
boolean found = false;
char ch;
int position = 0;
while (!found && position < str.length()) {
    ch = str.charAt(position);
    if (Character.isLowerCase(ch)) {
        found = true;
    }
    else { position++; }
}
```


Perguntar até que uma Ocorrência Seja Encontrada

- Inicialize uma variável sentinela/booleana com `false`
- Teste a variável sentinela no laço `while`
 - Leia a entrada, e compare com o limite
 - Se a entrada está dentro do limite, altere a variável sentinela para `true`
 - O laço parará de executar

```
boolean valido = false;
double entrada;
while (!valido) {
    System.out.print("Forneça um valor positivo < 100: ");
    entrada = in.nextDouble();
    if (0 < entrada && entrada < 100) { valido = true; }
    else { System.out.println("Entrada inválida!"); }
}
```

Máximo e Mínimo

- Leia o primeiro valor: este é o maior (ou menor) valor que você obteve até agora!
- Fique no laço enquanto você tiver um valor válido
 - Leia outro valor
 - Compare o novo valor com o maior (ou menor)
 - Atualize o maior (ou menor) valor se for necessário

```
double maior = in.nextDouble();
while (in.hasNextDouble()) {
    double valor = in.nextDouble();
    if (valor > maior) {
        maior = valor;
    }
}
```

```
double menor = in.nextDouble();
while (in.hasNextDouble()) {
    double valor = in.nextDouble();
    if (valor < menor) {
        menor = valor;
    }
}
```

Comparar Valores Adjacentes

- Obtenha o primeiro valor da entrada
- Use o `while` para determinar se há mais valores para serem verificados
 - Copie a entrada para uma variável para armazenar o valor anterior
 - Leia a próxima entrada
 - Compare a entrada lida com o valor anterior, e avise se forem iguais

```
double entrada = in.nextDouble();
while (in.hasNextDouble()) {
    double anterior = entrada;
    entrada = in.nextDouble();
    if (entrada == anterior) {
        System.out.println("Entrada duplicada!");
    }
}
```

Passos para Escrever um Laço

Planejamento:

- Decida que tarefa realizar dentro do laço
- Especifique a condição do laço
- Determine o tipo do laço
- Inicialize as variáveis antes da primeira iteração
- Processe os resultados depois que o laço tenha encerrado
- Teste o laço com exemplos típicos

Codificação:

- Implemente o laço em Java

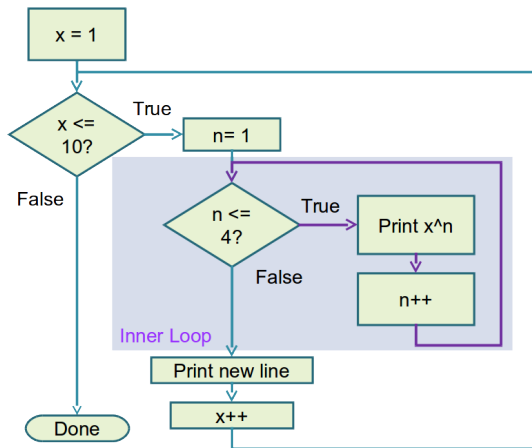
Laços Aninhados

Laços Aninhados

- Como você imprimiria uma tabela com linhas e colunas?
 - Imprima a primeira linha com o cabeçalho
 - Use um laço
 - Imprima o corpo da tabela
 - Quantas linhas?
 - Quantas colunas?
 - Faça um laço para as linhas
 - Faça um laço para as colunas

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
...
10	100	1000	10000

Fluxograma de Dois Laços Aninhados



TabelaDePotencias.java

```
/**
 * Este programa imprime uma tabela com as potências de x.
 * Adaptado de Horstmann (2013, p. 173-174).
 */
public class TabelaDePotencias {
    public static void main(String[] args) {
        final int NMAX = 4;
        final double XMAX = 10.0;
        // Imprime o cabeçalho da tabela
        for (int n = 1; n <= NMAX; n++) { System.out.printf("%10d", n); }
        System.out.println();
        for (int n = 1; n <= NMAX; n++) { System.out.printf("%10s", "x "); }
        System.out.println();
        // Imprime o corpo da tabela
        for (double x = 1; x <= XMAX; x++) {
            // Imprime uma linha
            for (int n = 1; n <= NMAX; n++) {
                System.out.printf("%10.0f", Math.pow(x, n));
            }
            System.out.println();
        }
    }
}
```


Resultado de TabelaDePotencias.java

1	2	3	4
x	x	x	x
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561
10	100	1000	10000

Exercício

Modifique o programa `TabelaDePotencias.java` para que a tabela seja impressa “deitada”, ou seja, na primeira linha os valores para x^1 , na segunda linha os valores para x^2 , e assim por diante.

Exemplos de Laços Aninhados (1)

Laço	Saída	Explicação
<pre>for (i = 1; i <= 3 ; i++) { for (j = 1; j <= 4; j++) { System.out.print("*"); } System.out.println(); }</pre>	<pre>**** **** ****</pre>	<p>Imprime 3 linhas de 4 asteriscos cada.</p>
<pre>for (i = 1; i <= 4 ; i++) { for (j = 1; j <= 3; j++) { System.out.print("*"); } System.out.println(); }</pre>	<pre>*** *** *** ***</pre>	<p>Imprime 4 linhas de 3 asteriscos cada.</p>

Exemplos de Laços Aninhados (2)

Laço	Saída	Explicação
<pre>for (i = 1; i <= 4 ; i++) { for (j = 1; j <= i; j++) { System.out.print("*"); } System.out.println(); }</pre>	<pre>* ** *** ****</pre>	<p>Imprime 4 linhas com tamanhos 1, 2, 3 e 4.</p>
<pre>for (i = 1; i <= 3 ; i++) { for (j = 1; j <= 5; j++) { if (j % 2 == 0) { System.out.print("*"); } else { System.out.print("-"); } } System.out.println(); }</pre>	<pre>-*-*- -*-*- -*-*-</pre>	<p>Imprime asteriscos nas colunas pares e traços nas colunas ímpares.</p>

Exemplos de Laços Aninhados (3)

Laço	Saída	Explicação
<pre>for (i = 1; i <= 3 ; i++) { for (j = 1; j <= 5; j++) { if (i % 2 == j % 2) { System.out.print("*"); } else { System.out.print(" "); } } System.out.println(); }</pre>	<pre>* * * * * * * *</pre>	Imprime o padrão de um tabuleiro de damas.

Exercício: faça laços aninhados para imprimir os seguintes padrões

a) *

```

.....
.*.....
..*.....
...*.....
....*.....
.....*.....
.....*
```

d)

```

.....*
.....*
.....*
.....*
.....*
.....*
```

g) *

```

*****
.******
..*****
...*****
....*****
.....*
```

j) *

```

*****
*.....*
*.....*
*.....*
*.....*
*.....*
*.....*
```

m) *

```

.....
**.....
*.*.....
*.*.....
*.*.....
*.*.....
*.....*
```

b) *

```

*****
.******
..*****
...*****
....*****
.....**
.....*
```

e) *

```

*****
*****.
*****.
*****.
*****.
***.....
**.....
*.....
```

h) *

```

*.....*
.*.....*
.*.....*
.*.....*
.*.....*
.*.....*
.*.....*
```

k)

```

.....
.....
.....
*****
.....
.....
.....*
```

n)

```

.....*
.....**
.....**
.....*
*.....*
*.....*
*.....*
```

c)

```

*.....
**.....
***.....
****.....
*****.
*****.
*****.
```

f)

```

.....*
.....**
.....***
.....***
.....***
*****.
*****.
```

i)

```

.....
.....
.....
.....*
.....*
.....*
*****
```

l) *

```

*.....*
.*.....*
.*.....*
*****
*****
*****
*.....*
```

o) *

```

*****
*.....*
*.....*
*.*.....
*.*.....
**.....
*.....*
```

Resumo

Resumo (1)

- Há 3 tipos de laços:
 - Laços `while`
 - Laços `for`
 - Laços `do`
- Cada laço possui as seguintes seções:
 - Inicialização (preparação das variáveis para iniciar o laço)
 - Condição (teste para verificar se o corpo do laço deve ser executado)
 - Atualização (alteração de alguma variável testada na condição)

Resumo (2)

- Um laço executa instruções repetidamente enquanto uma condição for verdadeira
- Errar o número de iterações em laço por uma unidade é um erro comum de programação
 - Procure deixar os testes simples para evitar este tipo de erro.
- O laço `for` é usado quando um valor varia de um ponto de partida até um ponto final com um incremento ou decremento constante
- O laço `do` é apropriado quando o corpo do laço deve ser executado pelo menos uma vez

Resumo (3)

- Um valor de sentinela consiste de um valor que determina o final de um conjunto de dados, mas que não faz parte deste conjunto
- Você pode usar uma variável booleana para controlar um laço
 - Defina a variável com `true` antes de entrar no laço, e então defina ela com `false` para sair do laço
- Quando o corpo de um laço contiver outro laço, os laços são aninhados
 - Um uso típico para laços aninhados é a impressão de uma tabela com linhas e colunas
- Em uma simulação, o computador é usado para simular uma atividade
 - Pode-se introduzir aleatoriedade chamando o gerador de números aleatórios

Tópicos Avançados

Tópicos Avançados

- Números Aleatórios e Simulações
- *Storyboards* para resolução de problemas
- Gráficos em Java

Números Aleatórios e Simulações

- Jogos frequentemente usam números aleatórios para tornar as coisas mais interessantes
 - Jogar dados
 - Girar uma roda
 - “Comprar” uma carta
- Uma simulação usualmente envolve executar um laço para uma sequência de eventos
 - Dias
 - Eventos

Números Aleatórios e Simulações: RandomDemo.java

- `Math.random()` pode ser usado para gerar números aleatórios no intervalo `[0;1)`

```
/**  
    Este programa imprime 10 números aleatórios entre 0 e 1.  
    Adaptado de Horstmann (2013, p. 176).  
*/  
public class RandomDemo {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; i++) {  
            double r = Math.random();  
            System.out.printf("%.20f\n", r);  
        }  
    }  
}
```

Resultado:

```
0,30625248942272576000  
0,57255173336825820000  
0,38359344863509290000  
0,67094454250114250000  
0,46419930546834340000  
0,45408986890317440000  
0,18747367004920823000  
0,36019379208793400000  
0,02052469620947972000  
0,59019359111533320000
```

Números Aleatórios e Simulações: lançamento de dados

- Pode-se simular o lançamento de dados usando `Math.random()`

```
/**
 * Este programa simula o lançamento de um par de dados.
 * Adaptado de Horstmann (2013, p. 177).
 */
public class Dados1 {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            // Gera números aleatórios entre 1 e 6
            // usando Math.random()
            int d1 = (int) (Math.random() * 6) + 1;
            int d2 = (int) (Math.random() * 6) + 1;
            System.out.println(d1 + " " + d2);
        }
    }
}
```

Resultado:

```
5 1
2 1
1 2
5 1
1 2
6 4
4 4
6 1
6 3
5 2
```

Números Aleatórios e Simulações: Dados2.java

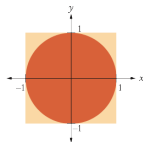
- Também é possível usar a classe `Random`, que tem muitos métodos para gerar números aleatórios em diferentes formatos

```
import java.util.Random;

/** Este programa simula o lançamento de um par de dados. */
public class Dados2 {
    public static void main(String[] args) {
        Random r = new Random();
        for (int i = 1; i <= 10; i++) {
            // Gera números aleatórios entre 1 e 6
            // usando o método nextInt() da classe Random
            int d1 = r.nextInt(6)+1;
            int d2 = r.nextInt(6)+1;
            System.out.println(d1 + " " + d2);
        }
    }
}
```


Números Aleatórios e Simulações: o método de Monte Carlo

- Usado para encontrar soluções aproximadas para problemas que não podem ser resolvidos com precisão
- Exemplo: aproximar o valor de π usando áreas relativas de um círculo dentro de um quadrado
 - Usa aritmética simples
 - Acertos estão dentro do círculo
 - Lançamentos são o número total de tentativas
 - Razão é $4 \times \text{Acertos} / \text{Lançamentos}$



Números Aleatórios e Simulações: MonteCarlo.java

```
/**
 * Este programa calcula uma estimativa do valor de pi
 * simulando o lançamento de dados em um quadrado.
 * Adaptado de Horstmann (2013, p. 178-179).
 */
public class MonteCarlo {
    public static void main(String[] args) {
        final int LANCAMENTOS = 10000;
        int acertos = 0;
        for (int i = 1; i <= LANCAMENTOS; i++) {
            // Gera dois números aleatórios entre -1 e 1
            double x = -1 + 2 * Math.random();
            double y = -1 + 2 * Math.random();
            // Verifica se (x,y) estão dentro do círculo unitário
            if (x * x + y * y <= 1)
                acertos++;
        }
        // A razão acertos / LANCAMENTOS é aproximadamente igual a
        // área do círculo / área do quadrado, que é pi/4
        double piEstimado = 4.0 * acertos / LANCAMENTOS;
        System.out.println("Estimativa de pi: " + piEstimado);
    }
}
```

Resultado:

Estimativa de pi: 3.1452

Storyboards para resolução de problemas

- Um storyboard (esboço sequencial) consiste numa sequência de desenhos anotados para cada etapa de uma sequência de ações
- Trata-se de uma técnica útil para solução de problemas que permite modelar a interação com o usuário
- Pode ajudar a responder:
 - Qual informação o usuário deve fornecer e em que ordem?
 - Qual informação o programa deve mostrar e em que formato?
 - O que deve acontecer se houver um erro?
 - Quando o programa deve terminar?

Storyboards: exemplo

- Objetivo: converter uma sequência de medidas
 - Exigirá um laço e algumas variáveis
 - Deverá gerenciar uma conversão de cada vez através de um laço

Converting a Sequence of Values

What unit do you want to convert from? **cm**

What unit do you want to convert to? **in**

Enter values, terminated by zero ————— Allows conversion of multiple values

30

30 cm = 11.81 in

100

100 cm = 39.37 in

0

What unit do you want to convert from?

Format makes clear what got converted

Storyboards: o que pode dar errado?

- Unidades de medidas desconhecidas
 - Como centímetros e polegadas são digitados?
 - Que outras conversões estão disponíveis?
- Solução: mostra uma lista de tipos de unidades aceitáveis

From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): **cm**
To unit: **in** — No need to list the units again

Storyboards: o que mais pode dar errado?

- Como o usuário encerra o programa?

Exiting the Program

From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): **cm**

To unit: **in**

Enter values, terminated by zero

30

30 cm = 11.81 in

0

More conversions (y, n)? **n**

(Program exits)

Sentinel triggers the prompt to exit

- Storyboards ajudam a planejar um programa: conhecer os fluxos ajuda a estruturar o código

Gráficos em Java

- Na sequência aparecem algumas dicas sobre como criar programas em Java que utilizam formas geométricas básicas
- O objetivo é apenas ter um programa com uma estrutura simples a partir da qual se possa desenhar algumas formas geométricas
- NÃO se pretende aprofundar a discussão sobre as classes usadas para criar e controlar janelas em uma *Graphic User Interface* (GUI)

Gráficos em Java: no método `main`

- Inicia-se criando um objeto chamado `frame` da classe `JFrame`
- Um `JFrame` corresponde a uma moldura dentro da qual se podem colocar ou desenhar outros componentes (no exemplo a seguir, a moldura ou janela será de 400 por 400 *pixels*)
- Para este *frame*, define-se então o tamanho (chamada ao método `setSize`) e a operação de fechamento padrão (chamada ao método `setDefaultCloseOperation`)
- Em seguida cria-se um componente (`JComponent`), definindo para este componente um método para desenhar a janela (basicamente este método, que se chama `paintComponent`, chama o método `draw`, que será responsável por desenhar a janela)
- Adiciona-se o componente ao *frame* (chamada de método `add`)
- E, por fim, torna-se o *frame* *visível* (chamada de método `setVisible`)

Gráficos em Java: no método `draw`

- É este método que efetivamente desenha as figuras geométricas na janela (e deve ser declarado como `static` para poder ser chamado a partir de `main`)
- O método `draw` tem como parâmetro um objeto da classe `Graphics`
- A classe `Graphics` possui uma série de métodos com os quais se pode desenhar diferentes figuras geométricas (retângulos, elipses, segmentos de reta, etc.)
- Pode-se considerar que os objetos da classe `Graphics` funcionam de forma semelhante a `System.out`, porém desenhando figuras em um *frame* e não textos em um terminal
- No exemplo a seguir, o método `setColor` define a cor padrão de impressão como sendo azul, e `fillRect` é usado para desenhar duas fileiras com quadrados preenchidos de forma alternada

Gráficos em Java: exemplo

- O exemplo a seguir desenha duas fileiras de retângulos alternados em uma janela






Gráficos em Java: LinhasDeQuadrados.java [Adaptado de Horstmann (2013, p. 180-181)]

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JComponent;




/** Este programa desenha duas linhas de quadrados. */
public class LinhasDeQuadrados {
    public static void draw(Graphics g) {
        final int TAMANHO = 20;
        g.setColor(Color.BLUE);
        int x = 0, y = 0;           // Linha do topo. O canto esquerdo superior tem as coordenadas (0,0)
        for (int i = 0; i < 10; i++) { g.fillRect(x + 2 * TAMANHO * i, y, TAMANHO, TAMANHO); }
        x = TAMANHO; y = TAMANHO; // Segunda linha, com deslocamento a partir da primeira
        for (int i = 0; i < 10; i++) { g.fillRect(x + 2 * TAMANHO * i, y, TAMANHO, TAMANHO); }
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame();           // O código que implementa o desenho está no método draw
        final int FRAME_WIDTH = 400, FRAME_HEIGHT = 400;
        frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent component = new JComponent() { public void paintComponent(Graphics graph) { draw(graph); } };
        frame.add(component);
        frame.setVisible(true);
    }
}
```

Gráficos em Java: alguns métodos de Graphics (1)

Método	Resultado	Explicação
<code>g.drawRect(x, y, width, height)</code>		(x, y) é o canto superior esquerdo.
<code>g.drawOval(x, y, width, height)</code>		(x, y) é o canto superior esquerdo do retângulo que limita a elipse. Para desenhar um círculo usa-se o mesmo valor para <code>width</code> e <code>height</code> .
<code>g.fillRect(x, y, width, height)</code>		O retângulo é desenhado preenchido.

Gráficos em Java: alguns métodos de Graphics (2)

Método	Resultado	Explicação
<code>g.fillOval(x, y, width, height)</code>		A elipse é desenhada preenchida.
<code>g.drawLine(x1, y1, x2, y2)</code>		$(x1, y1)$ e $(x2, y2)$ são os pontos inicial e final de um segmento de reta.
<code>g.drawString("Message", x, y)</code>		(x, y) é o ponto base (<i>basepoint</i>).

Gráficos em Java: alguns métodos de `Graphics` (3)

Método	Resultado	Explicação
<code>g.setColor(color)</code>	A partir deste ponto, os métodos para desenhar ou desenhar preenchido usarão a cor selecionada.	Use <code>Color.RED</code> , <code>Color.GREEN</code> , <code>Color.BLUE</code> e assim por diante.

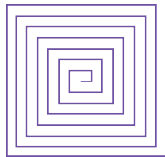
Gráficos em Java: exercícios

- 1 Escreva uma aplicação gráfica em Java para desenhar a seguinte face:



Fonte: Horstmann (2013, p. 197)

- 2 Escreva uma aplicação gráfica em Java para desenhar uma espiral retangular como a mostrada na figura a seguir:



Fonte: Horstmann (2013, p. 197)

Referências

Referências

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