

Métodos

Roland Teodorowitsch

Fundamentos de Programação - Escola Politécnica - PUCRS

15 de maio de 2023

Introdução

Objetivos

- Ser capaz de implementar métodos
- Tornar-se familiar com o conceito de passagem de parâmetros
- Desenvolver estratégias para decomposição de tarefas complexas em tarefas mais simples
- Ser capaz de determinar o escopo de variáveis
- Aprender como pensar recursivamente

Conteúdos

- Definição
- Implementando Métodos
- Passagem de Parâmetros
- Retorno de Valores
- Métodos sem Retorno de Valores
- Exercícios
- Reutilização de Métodos
- Escopo de Variáveis
- Métodos Recursivos
- Tópicos Avançados

Definição

O que é um Método?

- Um método é uma sequência de instruções com um nome
- Trata-se de um bloco de código que executa determinada tarefa e que foi isolado e identificado
- Desta forma, um método pode ser usado em muitos pontos do programa
 - Você declara um método definindo um bloco de código com nome

```
public static void main(String[] args) {  
    double resultado = Math.pow(2, 3);  
    // ...  
}
```

- Você chama o método de forma a executar as suas instruções
- Um método empacota uma computação formada por múltiplos passos em uma forma que pode ser facilmente entendida e reutilizada

Métodos

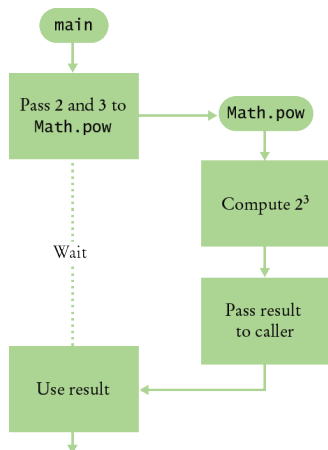
- Você já utilizou alguns métodos:

- `Math.pow()`
- `String.length()`
- `Character.isDigit()`
- `Scanner.nextInt()`
- `main()`

- Eles têm:

- Um nome de método: seguindo as mesmas regras de nomes de variáveis (estilo camelHump)
- Um par de parênteses ao seu final, para fornecer informação (argumentos) para a sua execução

Fluxograma da Chamada de um Método



```
public static void main(String[] args) {  
    double resultado = Math.pow(2, 3);  
    ...  
}
```

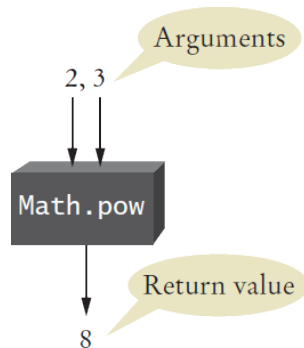
• Um método “chama” outro

- `main` chama `Math.pow()`
- Dois argumentos são passados: 2 e 3
- `Math.pow` inicia
 - Usa os argumentos (2, 3)
 - Faz o seu trabalho
 - Retorna a resposta
- `main` usa o resultado

Valores de Argumentos e Retorno

```
public static void main(String[] args) {  
    double resultado = Math.pow(2,3);  
    // ...  
}
```

- main “passa” dois argumentos (2 e 3) para `Math.pow`
- `Math.pow` calcula e retorna o valor 8 para main
- main armazena o valor de retorno na variável `resultado`



Analogia com uma Caixa Preta

- Um termostato é uma “caixa preta”
 - Defina a temperatura desejada
 - O termostato liga o aquecimento conforme necessário
 - Não é preciso saber como ele realmente funciona!
 - Como ele sabe a temperatura corrente?
 - Quais sinais/comandos ele deve enviar ao aquecedor para ligá-lo?
- Use métodos como “caixas pretas”
 - Passe para o método o que ele precisa para fazer o seu trabalho
 - Receba a resposta

Implementando Métodos

Implementando Métodos

- Um método para calcular o volume de um cubo
 - O que ele precisa para fazer os seu processamento?
 - Qual resposta ele retorna?
- Quando se escreve um método:
 - Escolha um nome para o método (`volumeCubo`)
 - Declare uma variável para cada argumento de entrada (`double lado`)
 - Determine o tipo do valor retornado (`double`)
 - Adicione modificadores tais como `public static`

```
public static double volumeCubo(double lado)
```

Dentro do Método

- Escreva o corpo do método
 - O corpo do método é colocado entre chaves
 - O corpo contém declarações de variáveis e comandos que são executados quando o método é chamado
 - O método também deve retornar o valor calculado

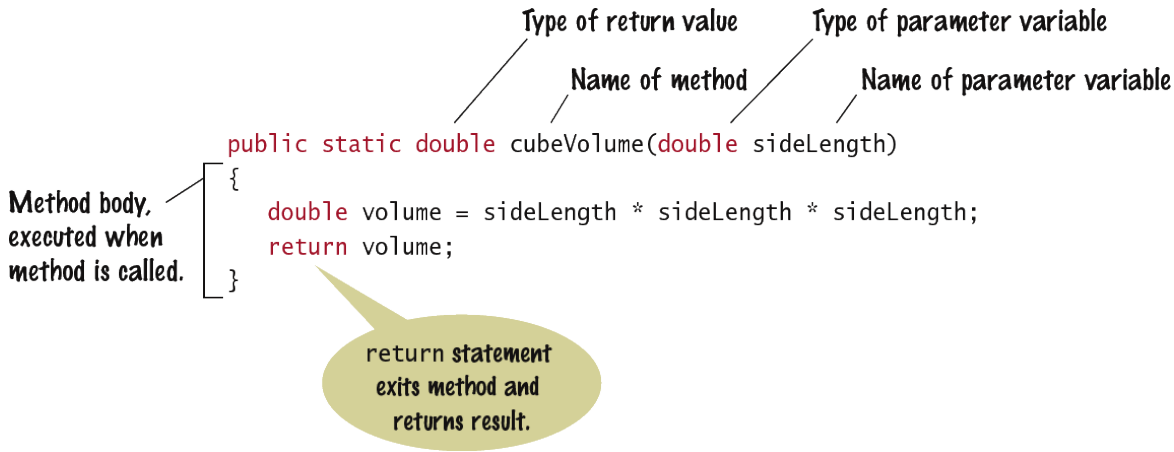
```
public static double volumeCubo(double lado) {  
    double volume = lado * lado * lado;  
    return volume;  
}
```

Fora do Método

- Os valores retornados por `volumeCubo` são armazenados em variáveis locais dentro de `main`
- Os resultados são mostrados

```
public static void main(String[] args) {  
    double lado1 = 2.0, volume1 = volumeCubo(lado1);  
    double lado2 = 10.0, volume2 = volumeCubo(lado2);  
    System.out.printf("Um cubo com lado %.1f tem volume %.1f.\n", lado1, volume1);  
    System.out.printf("Um cubo com lado %.1f tem volume %.1f.\n", lado2, volume2);  
}
```

Sintaxe da Declaração de Métodos



Comentários de Métodos

- Escreva um comentário Javadoc acima de cada método
- Inicie com `/**`
 - Indique o propósito do método
 - Descreva cada argumento de entrada com `@param`
 - Descreva o valor de retorno com `@return`
- Termine com `*/`

```
/**  
    Calcula o volume de um cubo.  
    @param lado Comprimento do lado do cubo.  
    @return O volume do cubo.  
*/  
public static double volumeCubo(double lado) {
```


Cubos.java

```
/** Este programa calcula o volume de dois cubos. Adaptado de: Horstmann (2013, p. 206). */
public class Cubos {

    /** Calcula o volume de um cubo.
     * @param lado Comprimento do lado do cubo.
     * @return O volume do cubo. */
    public static double volumeCubo(double lado) {
        double volume = lado * lado * lado;
        return volume;
    }

    /** Programa principal para realizar o cálculo do volume de dois cubos.
     * @param args Argumentos da linha de comando (NÃO utilizados). */
    public static void main(String[] args) {
        double lado1 = 2.0, volume1 = volumeCubo(lado1);
        double lado2 = 10.0, volume2 = volumeCubo(lado2);
        System.out.printf("Um cubo com lado %.1f tem volume %.1f.\n", lado1, volume1);
        System.out.printf("Um cubo com lado %.1f tem volume %.1f.\n", lado2, volume2);
    }
}
```

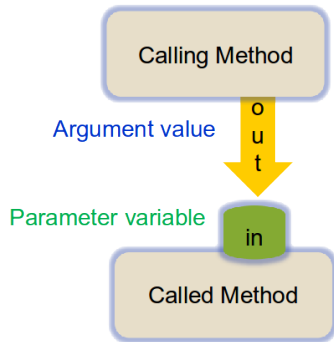
Resultado:

```
Um cubo com lado 2,0 tem volume 8,0.
Um cubo com lado 10,0 tem volume 1000,0.
```

Passagem de Parâmetros

Passagem de Parâmetros

- Variáveis de parâmetros recebem os valores dos argumentos fornecidos na chamada do método
 - Ambos devem ser do mesmo tipo
- O valor do **argumento** (parâmetro real) pode ser:
 - O conteúdo de uma variável
 - Um valor constante
 - Uma expressão
- **Variáveis de parâmetros** (parâmetros formais) são
 - Declaradas no método chamado
 - Inicializadas com o valor do argumento
 - Usadas como variáveis dentro do método chamado



Passos da Passagem de Parâmetros

```
public static void main(String[] args) {  
    double resultado1 = volumeCubo(2);  
    // ...  
}
```

sideLength = 2

volume = 8

```
public static double volumeCubo(double lado) {  
    double volume = lado * lado * lado;  
    return volume;  
}
```

result1 = 8

```
public static void main(String[] args) {  
    double resultado1 = volumeCubo(2);  
    // ...  
}
```

Erro Comum

Erro comum = tentar modificar argumentos

- Uma cópia do argumento é passada para o método
- O método chamado pode modificar cópias locais dos parâmetros, mas não pode modificar o valor original

```
public static void main(String[] args) {  
    double total = 10;  
    adicionaJuros(total, 7.5);  
}  
  
public static double adicionaJuros(double valor, double taxa) {  
    double juros = valor * taxa / 100;  
    preco = preco + juros; // NAO tem efeito fora do metodo  
    return juros;  
}
```

Retorno de Valores

Retorno de Valores

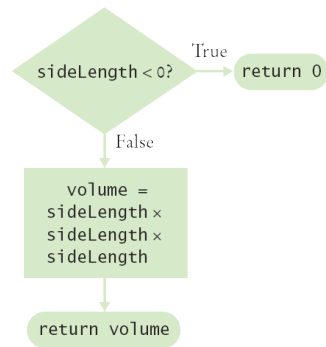
- Métodos podem (opcionalmente) retornar um valor
 - Declare um tipo de retorno na declaração do método
 - Adicione um comando `return` que retorna um valor
 - O comando `return` faz duas coisas
 - Termina imediatamente o método
 - Passa o valor de retorno de volta para a chamada do método
 - O valor de retorno pode ser uma constante, uma variável ou o cálculo de uma expressão, mas precisa corresponder ao tipo de retorno declarado

```
public static double volumeCubo(double lado) {  
    double volume = lado * lado * lado;  
    return volume;  
}
```

Múltiplos Comandos `return`

- Um método pode usar múltiplos comandos `return`
- Cada ramo de execução deve possuir um comando `return`

```
public static double volumeCubo(double lado) {  
    if (lado < 0) {  
        return 0;  
    }  
    return lado * lado * lado;  
}
```



Erro Comum

Erro comum = esquecer o comando `return`

- Verifique se todas as condições estão sendo tratadas
- No exemplo abaixo, `x` poderia ser igual a 0 e não foi previsto um `return` para esta situação
- O compilador vai reclamar se algum `return` for esquecido

```
public static int sign(double x) {  
    if (x < 0) { return -1; }  
    if (x > 0) { return 1; }  
    // Erro: falta um valor a ser retornado se x for igual a 0  
}
```

Implementando um Método: Passos

- 1 Descreva o que o método deveria fazer
- 2 Determine as entradas para o método
- 3 Determine os tipos dos parâmetros e do valor de retorno
- 4 Escreva pseudocódigo para obter o resultado desejado
- 5 Implemente o corpo do método

```
public static double pyramidVolume(double height, double baseLength)
{
    double baseArea = baseLength * baseLength;
    return height * baseArea / 3;
}
```

- 6 Teste o seu método: projete casos de teste

Métodos sem Retorno de Valores

Métodos sem Retorno de Valores

- Métodos não necessitam retornar um valor
- O tipo de retorno `void` significa que nada será retornado
- Nenhum comando `return` é necessário
- O método pode gerar saída!

```
public static void boxString(String str) {  
    int n = str.length();  
    for (int i = 0; i < n + 2; i++) { System.out.print("-"); }  
    System.out.println();  
    System.out.println("!" + str + "!");  
    for (int i = 0; i < n + 2; i++) { System.out.print("-"); }  
    System.out.println();  
}
```

```
...  
boxString("Hello");  
...
```

```
-----  
!Hello!  
-----
```

Usando `return` sem nenhum valor

- Você pode usar o comando `return` sem nenhum valor
- Em métodos com tipo de retorno `void`, o comando `return` encerra imediatamente o método

```
public static void boxString(String str) {  
    int n = str.length();  
    if (n == 0) {  
        return; // Return immediately  
    }  
    for (int i = 0; i < n + 2; i++) { System.out.print("-"); }  
    System.out.println();  
    System.out.println("!" + str + "!");  
    for (int i = 0; i < n + 2; i++) { System.out.print("-"); }  
    System.out.println();  
}
```

Exercícios

Exercícios

Implemente métodos para os seguintes casos:

- calcular as áreas de: quadrado, retângulo, círculo, trapézio, triângulo
- verificar se, dados 3 comprimentos, eles poderiam corresponder aos lados de um triângulo ou não
- calcular o fatorial de um número
- verificar se um número é primo ou não
- determinar se uma cadeia de caracteres é um palíndromo ou não (exemplos: "OVO"é palíndromo, "CASA"não é)

Reutilização de Métodos

Reutilização de Métodos

- Procure código “repetido”
 - Pode envolver valores diferentes, porém com a mesma lógica

```
Scanner in = new Scanner(System.in);  
int horas;  
do {  
    System.out.print("Digite um valor de 0 a 23: ");  
    horas = in.nextInt();  
} while (horas < 0 || horas > 23);  
  
int minutos;  
do {  
    System.out.print("Digite um valor de 0 a 59: ");  
    minutos = in.nextInt();  
} while (minutos < 0 || minutos > 59);
```

Escreva um método parametrizado

```
/**
 Solicita que o usuario digite um valor, aceitando apenas
 valores em um intervalo predefinido.

 @param inf o limite inferior do intervalo
 @param sup o limite superior do intervalo
 @return o valor digitado pelo usuario
 */
public static int leiaValoresNoIntervalo(Scanner in, int inf, int sup) {
    int entrada;
    do {
        System.out.print("Digite um valor de " + inf + " a " + sup + ": ");
        entrada = in.nextInt();
    } while (entrada < inf || entrada > sup);
    return entrada;
}

/* ... */

Scanner in = new Scanner(System.in);
int horas = leiaValoresNoIntervalo(in, 0, 23);
int minutos = leiaValoresNoIntervalo(in, 0, 59);
```

Escopo de Variáveis

Escopo de Variáveis

- Variáveis podem ser declaradas
 - Dentro de um método
 - Chamadas de “variáveis locais” (ou de escopo local)
 - Somente podem ser acessadas dentro do método
 - Variáveis paramétricas são deste tipo
 - Dentro de um bloco de código `{ }`
 - Chamadas de “variáveis de bloco” (ou de escopo de bloco)
 - Não podem ser acessadas depois do fim do bloco
 - Fora de um método
 - Chamadas de “variáveis globais” (ou de escopo global)
 - Podem ser usadas (e alteradas) por código em qualquer método
- O escopo de uma variável corresponde à parte do programa onde ela é visível
- Como escolher? Qual a melhor forma?

Exemplo de Escopo

- soma é uma variável local em main
- quadrado é visível apenas dentro do bloco do laço for
- i é visível apenas dentro do laço for

```
public static void main(String[] args) { // ESCOPOS:
    int soma = 0;                        // soma
    for (int i = 1; i <= 10; i++) {      // soma, i
        int quadrado = i * i;           // soma, i, quadrado
        soma = soma + quadrado;         // soma, i, quadrado
    }                                    // soma
    System.out.println(soma);           // soma
}
```

Variáveis Locais de Métodos

- Variáveis declaradas dentro de um método **não** são visíveis a outros métodos
 - lado é local do método main
 - Usá-la fora do método causará um erro de compilação

```
public static void main(String[] args) {  
    double lado = 10;  
    double resultado = volumeCubo();  
    System.out.println(resultado);  
}  
  
public static double volumeCubo() {  
    return lado * lado * lado; // ERRO!  
}
```

Reutilização de Nomes de Variáveis Locais

- Variáveis declaradas dentro de um método **não** são visíveis a outros métodos
 - resultado é local de `quadrado` e resultado também é local em `main`
 - São duas variáveis diferentes que não se sobrepõem

```
public static int quadrado(int n) {  
    int resultado = n * n;  
    return resultado;  
}  
  
public static void main(String[] args) {  
    int resultado = quadrado(3) + quadrado(4);  
    System.out.println(resultado);  
}
```

Reutilização de Nomes de Variáveis de Blocos

- Variáveis declaradas dentro de um bloco **não** são visíveis a outros blocos
 - `i` está dentro do primeiro bloco e `i` também está dentro do segundo bloco
 - São duas variáveis diferentes que não se sobrepõem

```
public static void main(String[] args) {  
    int soma = 0;  
    for (int i = 1; i <= 10; i++) {  
        soma = soma + i;  
    }  
    for (int i = 1; i <= 10; i++) {  
        soma = soma + i * i;  
    }  
    System.out.println(soma);  
}
```


Escopos Sobrepostos

- Variáveis (incluindo variáveis paramétricas) devem ter nomes únicos dentro de seu escopo
 - n tem escopo local e n está declarado em um bloco dentro deste escopo
 - O compilador vai reclamar quando a variável n for declarada no escopo do bloco

```
public static int somaDeQuadrados(int n) { // ESCOPOS:
    int soma = 0;                          // n
    for (int i = 1; i <= n; i++) {          // n
        int n = i * i; /* ERRO */           // n, n
        soma = soma + n;                    // n, n
    }                                       // n
    return soma;                           // n
}
```

Sobreposição Global e Local

- Variáveis globais e locais (de métodos) podem se sobrepor
 - A variável local `mesma` será usada no seu contexto local
 - Não há acesso à variável global `mesma` quando `mesma` local estiver em seu escopo

```

public class Escopo {                                     // ESCOPOS:
    public static int mesma; /* global */                 // mesma
    public static void main(String[] args) {             // mesma
        int mesma = 0; /* local */                       // mesma, mesma
        for (int i = 1; i <= 10; i++) {                  // mesma, mesma
            int quadrados = i * i;                       // mesma, mesma
            mesma = mesma + quadrados;                    // mesma, mesma
        }                                                 // mesma, mesma
        System.out.println(mesma);                       // mesma
    }
}

```

- Variáveis com o mesmo nome em diferentes escopos não geram erro de compilação, mas não é uma boa ideia fazer isto

Exercício 1

Considere o programa em Java a seguir e mostre o que será impresso, respeitando a ordem de execução.

```
public class Testel {  
    public static int a = 9;  
    public static int b = 21;  
  
    public static int metodo1(int b) {  
        b++;  
        System.out.println("[1] "+a);  
        System.out.println("[2] "+b);  
        int a = 5;  
        a *= 3;  
        System.out.println("[3] "+a);  
        return a;  
    }  
  
    public static int metodo2(int a) {  
        System.out.println("[4] "+a);  
        System.out.println("[5] "+b);  
        int b = (a%2==0)?1:0;  
        System.out.println("[6] "+b);  
        return b;  
    }  
}
```

```
public static void main(String[] args) {  
    a++;  
    b--;  
    System.out.println("[7] "+a);  
    System.out.println("[8] "+b);  
    int a = metodo1(b);  
    int b = metodo2(a);  
    System.out.println("[9] "+a);  
    System.out.println("[10] "+b);  
    System.out.println("[11] "+Testel.a);  
    System.out.println("[12] "+Testel.b);  
}
```

Exercício 2

Indique o escopo de cada uma das variáveis do programa abaixo. Determine o que o programa imprimirá, sem executá-lo no computador (HORSTMANN, 2013, p. 237).

```
public class Exemplo {  
  
    public static void main(String[] args) {  
        int i = 10;  
        int b = g(i);  
        System.out.println(b + i);  
    }  
  
    public static int f(int i) {  
        int n = 0;  
        while (n * n <= i) { n++; }  
        return n - 1;  
    }  
}
```

```
    public static int g(int a) {  
        int b = 0;  
        for (int n = 0; n < a; n++) {  
            int i = f(n);  
            b = b + i;  
        }  
        return b;  
    }  
}
```

Métodos Recursivos

Métodos Recursivos

- Um método recursivo é um método que chama a si mesmo
- Uma computação recursiva resolve um problema dividindo o problema em computações mais simples e aplicando essa mesma estratégia sobre estas computações mais simples
- Para uma recursão terminar, deve haver pelo menos um caso especial para essas entradas mais simples
- Métodos recursivos são comuns na matemática - o fatorial é um exemplo:
 - Fatorial de 0 e de 1 é por definição 1: $0! = 1$ e $1! = 1$
 - Fatorial de n é igual a n vezes o fatorial de $n - 1$: $n! = n \times (n - 1)!$

Exemplo: Fatorial (Versões NÃO Recursiva e Recursiva)

Exemplo: Fatorial (Versões NÃO Recursiva e Recursiva)

```
public class Fatorial {  
  
    public static long fatorial(int n) {  
        long res = 1;  
        for (int i=2; i<=n; ++i) res = res * i;  
        return res;  
    }  
  
    public static long fatorialRec(int n) {  
        if (n <= 1) return 1L;  
        return n * fatorialRec(n-1);  
    }  
  
    public static void main(String[] args) {  
        System.out.printf("%d! = %d\n", 5, fatorial(5) );  
        System.out.printf("%d! = %d\n", 5, fatorialRec(5) );  
    }  
}
```


Exemplo: Triângulo Recursivo

- O método chama a si mesmo (e não mostra nada) até que `tam` seja menor do que 1
- Neste caso, ele usa o comando `return` e todas as iterações anteriores imprimem os seus resultados: 1, 2, 3, 4

```
public class Triangulo {

    public static void triangulo(int tam) {
        if (tam < 1) return;
        triangulo(tam - 1);
        for (int i = 0; i < tam; i++)
            System.out.print("*");
        System.out.println();
    }

    public static void main(String[] args) {
        triangulo(4);
    }

}
```

- Aqui está o que acontece quando se manda imprimir um triângulo com tamanho igual a 4:
 - A chamada `triangulo(4)` chama `triangulo(3)`
 - A chamada `triangulo(3)` chama `triangulo(2)`
 - A chamada `triangulo(2)` chama `triangulo(1)`
 - A chamada `triangulo(1)` chama `triangulo(0)`
 - A chamada `triangulo(0)` retorna, sem fazer nada
 - A chamada `triangulo(1)` imprime *
 - A chamada `triangulo(2)` imprime **
 - A chamada `triangulo(3)` imprime ***
 - A chamada `triangulo(4)` imprime ****

Recursão

- Quando chamamos um método, o código que fez a chamada fica esperando até a chamada do método seja concluída
- Na execução de um método recursivo, acontece a mesma coisa
 - Temos instâncias do método esperando que outras instâncias do mesmo método terminem
 - A chamada que foi executada antes aguarda que a chamada que ela fez termine - e isso acontece de forma encadeada
- Em algum momento, o método recursivo precisa parar de se invocar, pois senão teríamos uma espécie de laço infinito
- Para evitar problemas de inconsistência, métodos recursivos devem evitar acessar variáveis globais
- De forma geral, se temos um problema resolvido com laços aninhados, a versão recursiva consegue eliminar um nível de laços

Exercícios (1-2)

- 1 O número de Fibonacci de ordem n pode ser definido como a soma dos dois números de Fibonacci anteriores. Ou seja:

$\text{Fibonacci}(n) = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$

e, por definição, $\text{Fibonacci}(1) = 0$ e $\text{Fibonacci}(2) = 1$.

Implemente, em Java:

- Um método NÃO recursivo que recebe n , calculando e retornando o número de Fibonacci de ordem n ;
 - Um método recursivo que recebe n , calculando e retornando o número de Fibonacci de ordem n .
- 2 Considerando a implementação recursiva dos números de Fibonacci, caso seja necessário usar esse método para calcular o número de Fibonacci de ordem 7, identifique:
- Quantas vezes o método fibonacci recursivo será chamado?
 - Quantas vezes o método fibonacci recursivo será chamado para n valendo 5?

Exercícios (3-5)

- 3 Implemente um método recursivo em Java que recebe n (inteiro e maior ou igual a 1) e que imprima os números inteiros de 1 até n , inclusive.
- 4 Implemente um método recursivo em Java que recebe n (inteiro e maior ou igual a 1) e que imprima os números inteiros de n até 1, inclusive.
- 5 Dado um valor inteiro e positivo (n), o valor da constante de *Euler* poder ser calculando com precisão diretamente proporcional a n através da fórmula:

$$E = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$$

Implemente, em Java:

- Um método NÃO recursivo que recebe n , retornando o valor de *Euler* calculado usando a fórmula acima;
- Um método recursivo que recebe n , retornando o valor de *Euler* calculado usando a fórmula acima.

Exercício (6)

- 6 Considere o programa em Java a seguir e mostre o que será impresso, respeitando a ordem de execução.

```
public class Teste2 {  
    public static int a = 2;  
    public static int b = 10;  
  
    public static void metodo1(int b) {  
        System.out.println("[1] "+a);  
        int a = b;  
        System.out.println("[2] "+a);  
        if (b==1)  
            return;  
        else  
            metodo1(b-1);  
    }  
}
```

```
public static void metodo2() {  
    System.out.println("[3] "+a);  
    System.out.println("[4] "+b);  
    metodo1(a);  
    ++b;  
    int b = 2;  
    System.out.println("[5] "+a);  
    System.out.println("[6] "+b);  
}  
  
public static void main(String[] args) {  
    System.out.println("[7] "+a);  
    a++;  
    int a = 15;  
    System.out.println("[8] "+a);  
    a++;  
    metodo2();  
    System.out.println("[9] "+a);  
}
```

Sumário

Sumário: Métodos

- Um método é um nome para uma sequência de instruções
- Argumentos são fornecidos quando um método é chamado. O valor de retorno é o resultado que o método computa
- Quando se declara um método, deve-se prover um nome para o método, uma variável para cada argumento, e um tipo para o resultado
- Comentários de métodos explicam o propósito do método, o significado dos parâmetros e o valor de retorno, bem como qualquer requisito especial
 - Sempre escreva uma descrição de cada variável paramétrica e do valor de retorno
- Variáveis paramétricas armazenam os argumentos fornecidos na chamada do método

Sumário: Retornos de Métodos

- O comando `return` termina a execução de um método e especifica o resultado calculado pelo método
 - Coloque computações que podem ser necessárias mais de uma vez dentro de métodos
 - Use o tipo `void` para indicar que o método não retorna nenhum valor
- Use o processo de refinamento passo-a-passo para decompor tarefas complexas em tarefas mais simples
 - Um método pode necessitar de outros métodos mais simples para executar o seu trabalho

Sumário: Escopo

- O escopo de uma variável é a parte do programa em que ela é visível
 - Duas variáveis locais ou paramétricas podem ter o mesmo nome desde que seus escopos não se sobreponham
 - Pode-se usar o mesmo nome de variável dentro de diferentes métodos desde que seus escopos não se sobreponham
 - Variáveis locais declaradas dentro de um método não são visíveis ao código dentro de outros métodos

Sumário: Recursão

- Uma computação recursiva soluciona um problema usando a solução do mesmo problema com entradas mais simples
 - Para uma recursão encerrar, deve haver um caso especial para as entradas mais simples
 - A chave para encontrar uma solução recursiva é reduzir a entrada para uma entrada mais simples para o mesmo problema
 - Quando se projeta uma solução recursiva, não se preocupe com múltiplas chamadas aninhadas. Simplesmente mantenha o foco em reduzir o problema para uma forma mais simples

Exercícios

Exercícios 1

1 Escreva os seguintes metodos em Java e teste-os em um método main():

- `double smallest(double x, double y, double z)`, retornando o menor valor entre todos os argumentos (por exemplo, `smallest(0, -1, 3.45)` retorna `-1.0000`)
- `double average(double x, double y, double z)`, retornando o valor medio de todos os argumentos (por exemplo, `average(1, 2, 6)` retorna `3.0000`)
- `boolean allTheSame(double x, double y, double z)`, retornando `true` se todos forem iguais, ou `false` em caso contrario (por exemplo, `allTheSame(1, 1, 1)` retorna `true` e `allTheSame(2, 2, 2.1)` retorna `false`)
- `boolean allDifferent(double x, double y, double z)`, retornando `true` se todos forem diferentes, ou `false` em caso contrario (por exemplo, `allDifferent(1, 2, 3)` retorna `true` e `allDifferent(1, 1, 2)` retorna `false`)
- `boolean sorted(double x, double y, double z)`, retornando `true` se os argumentos estiverem ordenados ($x \leq y \leq z$), ou `false` em caso contrario (por exemplo, `sorted(1, 3, 5)` retorna `true` e `sorted(1, 3, 2)` retorna `false`)
- `int firstDigit(int n)`, retornando o primeiro digito do argumento (por exemplo, `firstDigit(1729)` retorna `1`)
- `int lastDigit(int n)`, retornando o ultimo digito do argumento (por exemplo, `lastDigit(1729)` retorna `9`)
- `int digits(int n)`, retornando o numero de digitos do argumento (por exemplo, `digits(1729)` retorna `4`)

Exercício 2

- 2 Implemente em Java dois métodos, um não recursivo e outro recursivo, que recebam um valor inteiro n e retornem o valor real correspondente ao n -ésimo número harmônico. O n -ésimo número harmônico, $H(n)$, com n maior ou igual a 1, pode ser calculado da seguinte forma:

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$$

Tópicos Avançados

Tópicos Avançados

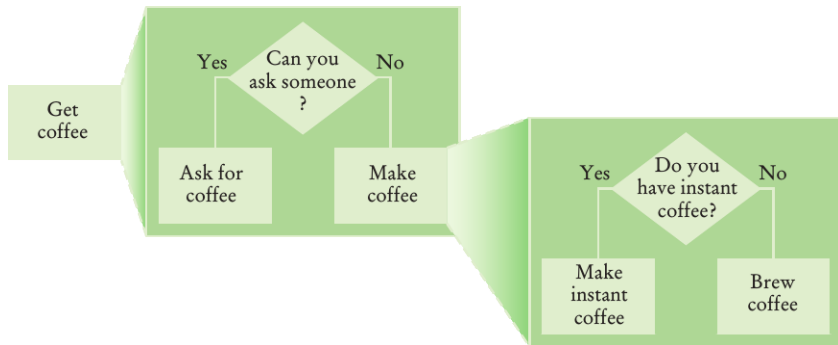
- Refinamento de passos

Refinamento de Passos

- Para resolver uma tarefa difícil, decomponha ela em tarefas mais simples
- Então continue decompondo as tarefas simples em tarefas cada vez mais simples, até que você chegue a tarefas que você saiba como resolver

Refinamento de Passos: Quer Tomar Café? (Exemplo)

- Se você tiver que fazer um café, há duas opções:
 - Faça um café instantâneo
 - Passe um café



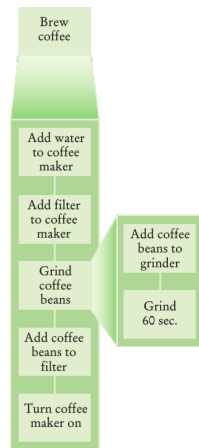
Refinamento de Passos: Café Instantâneo

- Há duas formas de ferver a água:
 - Use o microondas
 - Use uma chaleira no fogão



Refinamento de Passos: Café Passado

- Considerando o uso de uma cafeteira:
 - Adicione água
 - Adicione o filtro
 - Moa o café
 - Adicione os grãos no moedor
 - Moa por 60 segundos
 - Encha o filtro com o café moído
 - Ligue a máquina de café



Refinamento de Passos: Exemplo

- Quando se preenche um cheque, costuma-se preencher o valor total tanto no formato numérico (R\$274,15) quanto no formato textual (duzentos e setenta e quatro reais e quinze centavos). Escreva um programa para transformar um número em um texto.
- Mesmo que pareça difícil, pode-se iniciar fazendo algumas **simplificações** para a seguir **decompor** o problema
 - Inicialmente separe a parte inteira (valor em reais sem centavos) e considere apenas valores de 0 até 999
 - Analise a questão dígito por dígito: isole os dígitos para centenas, dezenas e unidades
 - Você vai precisar de um método para resolver o problema (por exemplo, `numeroPorExtenso`)
 - Este método poderá usar métodos para cada tipo de dígito: `nomeCentena`, `nomeDezena` e `nomeUnidade`
 - **Identifique e resolva as situações gerais, depois trate os casos específicos!**

Refinamento de Passos: Tratando Centenas

- Método `nomeCentena`:
 - Identifique todas as palavras envolvidas e também situações em que elas serão utilizadas
 - Se não houver centena, não há nada a fazer
 - Se a centena for igual a 1, há duas possibilidades
 - Sem resto, a palavra é “cem”
 - Havendo resto, o texto deverá ser “cento e ”
 - Para as outras centenas, os textos são bem comportados: “duzentos”, “trezentos”, “quatrocentos”, etc.
 - Havendo resto, após o texto deverá ser acrescentado “ e ”

Refinamento de Passos: Tratando Dezenas

- Método `nomeDezena`:

- Identifique todas as palavras envolvidas e também situações em que elas serão utilizadas
- Se não houver dezena, não há nada a fazer
- Se a dezena for igual a 1, dezena e resto devem ser tratados de forma unificada, então é melhor fazer isto no método das unidades
- Para as outras dezenas, os textos são bem comportados: “vinte”, “trinta”, “quarenta”, etc.
 - Havendo resto, após o texto deverá ser acrescentado “ e ”

Refinamento de Passos: Tratando Unidades

- Método `nomeUnidade`:

- Identifique todas as palavras envolvidas e também situações em que elas serão utilizadas
- Se não houver unidade, não há nada a fazer
- Se houver unidade, os textos são: “um”, “dois”, “tres”, “quatro”, etc.
- Este método também tratará as dezenas como se fossem unidades, portanto, também haverá textos para: “dez”, “onze”, “doze”, “treze”, “catorze”, etc.

Refinamento de Passos: Tratando o Zero

- 0 (“zero”) é um caso especial e deve ser tratado à parte

Refinamento de Passos: Escreva Pseudocódigo

```
numero = n (numero a ser convertido)
texto  = "" (texto correspondente ao numero)
se numero for 0 entao texto = "zero"
senao
inicio
  // calcula numero de centenas, dezenas e unidades
  centenas = numero / 100
  restoCentenas = numero % 100
  dezenas = restoCentenas / 10
  unidades = restoCentenas % 10

  // trata dezenas como unidades
  se dezenas for igual a 1
  entao inicio
    dezenas = 0
    unidades = unidades + 10
  fim

  // gera resultado
  texto = nomeCentena(centenas, restoCentenas) + nomeDezena(dezenas, unidades) + nomeUnidade(unidades)
fim
```

Refinamento de Passos: Planeje os Métodos

- `main` **chama** `numeroPorExtenso`
- `numeroPorExtenso` **faz todo o trabalho** retornando um `String`
- `numeroPorExtenso` **usa os métodos** `nomeCentena`, `nomeDezena` e `nomeUnidade`
- `nomeCentena` **recebe** o número de centenas e o resto, retornando um `String`
- `nomeDezena` **recebe** o número de dezenas e o resto, retornando um `String`
- `nomeUnidade` **recebe** o número de unidades, retornando um `String`

Refinamento de Passos: Dicas de Programação

- Mantenha os métodos pequenos: se ficarem maiores do que uma página, quebre eles em submétodos
- Execute seus métodos manualmente, mostrando a evolução da alteração das variáveis
 - Use linhas para cada passo e reserve colunas para as variáveis importantes

intName(number = 416)	
part	name
416	
16	"four hundred"
0	"four hundred sixteen"

- Use métodos incompletos, com conteúdo provisório, que retornam valores “fictícios” durante o desenvolvimento

```
public static String nomeUnidade(int unidades) {  
    return "unidade";  
}
```

Referências

Referências

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