

Programação I

Métodos estáticos e Estruturas de repetição

Samuel da Silva Feitosa

Aula 4

Métodos Estáticos

Introdução

- A modularização de código é algo essencial no desenvolvimento de software atualmente.
 - Procedimentos e funções são conceitos utilizados para este fim.
- A linguagem Java implementa ambos os conceitos através de métodos.
 - É possível separar as implementações em múltiplos arquivos.

O que já sabemos?

- É comum dividir o problema / algoritmo em diversos pedaços, cada qual com sua funcionalidade.
 - Daí vem o nome **modularização** ou **programação estruturada**.
- Já conhecemos os conceitos de **procedimentos e funções**.
 - Ambos são trechos de código separados da linha de execução principal, que podem ser **chamados** quantas vezes for necessário.

Procedimentos

- Um procedimento é uma sub-rotina que realiza uma operação e não retorna nenhum valor.

```
Procedimento Soma(a: Inteiro, b: Inteiro)
Var
    sm: Inteiro;
Início
    sm := a + b;
    Escreva("A soma é: ", sm);
Fim
```

Parâmetros

```
Soma(2, 3);
```

Chamada do procedimento

Funções

- Uma função é uma sub-rotina que realiza alguma operação e retorna algum valor.

```
Função Soma(a: Inteiro, b: Inteiro): Inteiro
```

```
Var
```

```
    sm: Inteiro;
```

```
Início
```

```
    sm := a + b;
```

```
    retorne sm;
```

```
Fim
```

```
res = Soma(2, 3);
```

```
Escreva("A soma é: ", res);
```

Tipo de retorno

Retorna a variável com o resultado

Variável resultado

Parâmetros e Escopo

- Informações e variáveis podem ser **passadas por parâmetro** através da chamada de funções ou procedimentos.
- Variáveis criadas **dentro de blocos** tem **escopo local**, ou seja, valem apenas dentro do bloco em que foram criadas.
 - Outras variáveis não podem ser utilizadas, a menos que sejam variáveis globais.

Vantagens da Modularização

- **Dividir e estruturar** um algoritmo em partes logicamente coerentes.
- Facilidade de testar trechos em separado.
- Evitar repetição do código-fonte.
- Maior legibilidade de código.

Uso na Linguagem Java

- A linguagem Java não possui conceitos separados para procedimentos e funções.
- Para implementar ambas as funcionalidades são utilizados os **métodos**.
- De forma similar, **métodos** são trechos de código que permitem realizar ações ou transformações sobre valores.

Métodos sem retorno

- Um método sem retorno possui o mesmo comportamento de um procedimento.
 - Para indicar que o método não tem retorno usamos a palavra-chave **void**.

```
static void soma(int a, int b) {  
    int sm = a + b;  
    System.out.println("A soma é: " + sm);  
}  
public static void main(String[] args) {  
    soma(2, 3);  
}
```

Métodos com retorno

- Um método com retorno possui a mesma funcionalidade de uma função.

```
static int soma(int a, int b) {  
    int sm = a + b;  
    return sm;  
}  
  
public static void main(String[] args) {  
    int res = soma(2, 3);  
    System.out.println("A soma é: " + res);  
}
```

Múltiplos arquivos

- Métodos em Java podem estar dispostos em múltiplos arquivos.
 - Essa divisão facilita a administração de código, deixando cada arquivo responsável por uma funcionalidade específica do sistema.
 - Permite também a reutilização de código, onde é possível realizar o compartilhamento apenas dos arquivos (bibliotecas) necessários para outros projetos.
- Vejamos alguns exemplos.

Classe `java.lang.Math`

- Classe que contém métodos estáticos para a realização de cálculos numéricos.
 - `E/PI`: representam as constantes de Euler (`e`) e `PI`.
 - `acos` / `asin` / `atan`, `cos` / `sin` / `tan`: relacionados ao cosseno, seno e tangente.
 - `ceil` / `floor` / `rint` / `round`: teto, piso, arredondamentos.
 - `sqrt` / `cbrt` / `pow`: raiz quadrada, cúbica e potência.
 - `log` / `log10`: logaritmos.
 - `abs`, `max` / `min`: valor absoluto, máximo e mínimo.
 - `random`: retorna um valor aleatório.

Laços de Repetição

Introdução

- Existem ocasiões que é necessário efetuar a **repetição de um trecho do programa** um determinado número de vezes.
- Os conceitos de loop ou laços de repetição permitem executar o processamento de um trecho de código quantas vezes forem necessárias.
- Java possui três tipos de laços de repetição.
 - **while, do...while, for.**

Laços While

- O tipo mais simples de laço em Java é o laço **while**.
 - Testa se uma certa condição é satisfeita e executa o corpo do laço enquanto essa condição for **true**.
- No início de cada **iteração**, o laço testa a expressão booleana.
 - Se for **true**, executa o corpo do laço.
 - Se for **false**, encerra a execução do laço.

```
while (expr_booleana) {  
    // corpo do laço  
}
```


Exemplo While com Número

- Calcular o quadrado de vários números informados pelo usuário. Zero é o critério de parada.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    System.out.println("Informe um número: ");  
    int num = sc.nextInt();  
  
    while (num > 0) {  
        int quadrado = num * num;  
        System.out.println("O quadrado de " + num + " é " + quadrado);  
  
        System.out.println("Informe outro número (0 para sair): ");  
        num = sc.nextInt();  
    }  
}
```

Exemplo While com String

- Fazer a repetição de uma mensagem enquanto o usuário informar 'S'. A letra 'N' é usada como critério de parada.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    System.out.println("Você está com fome? (S/N)");  
    String resp = sc.nextLine();  
  
    while (resp.toUpperCase().equals("S")) {  
        System.out.println("Prepare a mesa e coma um lanche!");  
  
        System.out.println("Você continua com fome? (S/N)");  
        resp = sc.nextLine();  
    }  
}
```

Contadores e Acumuladores

- Juntamente com as instruções de repetição, aparecem os conceitos de **contadores** e **acumuladores**.
 - **Contadores**: contam a quantidade de repetições de execução de um determinado laço. Bastante utilizado como critério de parada.
 - **Acumuladores**: acumulam um valor a cada passo de repetição do laço. Normalmente utilizado para cálculos.

Exemplo: Contadores e Acumuladores

- Lê o preço de vários produtos e calcula o total da compra. Zero é o critério de parada.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    System.out.println("Informe o preço do produto: (0 para sair)");  
    double preco = sc.nextDouble();  
  
    int contador = 0;  
    double acumulador = 0;  
  
    while (preco > 0) {  
        acumulador = acumulador + preco;  
        contador = contador + 1;  
  
        System.out.println("Informe o preço do produto: (0 para sair)");  
        preco = sc.nextDouble();  
    }  
  
    System.out.println("Foram informados " + contador + " produtos.");  
    System.out.println("Total da compra: " + acumulador);  
}
```

Laços For

- Na sua forma mais simples, os laços **for** oferecem uma repetição baseada em um índice inteiro.
- Sua estrutura se divide em quatro seções: inicialização, condição, incremento e corpo.
 - **Inicialização**: pode-se declarar uma variável índice.
 - **Condição**: expressão booleana que especifica o critério de parada do laço.
 - **Incremento**: permite incrementar o contador do laço.

```
for (inicialização; condição; incremento) {  
    // corpo do laço  
}
```

Exemplo For (1)

- Cálculo da média de notas de uma turma de 5 alunos.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    double acumulador = 0;  
  
    for (int i = 0; i < 5; i++) {  
        System.out.println("Informe a média do aluno: ");  
        double mediaAluno = sc.nextDouble();  
  
        acumulador = acumulador + mediaAluno;  
    }  
  
    double mediaTurma = acumulador / 5;  
    System.out.println("A média da turma é: " + mediaTurma);  
}
```

Exemplo For (2)

- Cálculo do fatorial de um número informado pelo usuário.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    System.out.println("Informe o número para calcular o fatorial: ");  
    int num = sc.nextInt();  
  
    int fatorial = 1;  
  
    for (int i = 1; i <= num; i++) {  
        fatorial = fatorial * i;  
    }  
    System.out.println("O fatorial de " + num + " é " + fatorial);  
}
```

Laços Do...While

- Os laços que vimos anteriormente testam a condição antes de executar a primeira iteração com o corpo do laço.
- O laço **do...while** testa a condição após o corpo do laço.
 - Dessa forma, o corpo é sempre executado ao menos uma vez.

```
do {  
    // corpo do laço  
} while (expr_booleana);
```


Exemplo Do...While

- Faz a leitura de nomes e mostra a quantidade de caracteres de cada um.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    String opcao;  
  
    do {  
        System.out.println("Informe um nome: ");  
        String nome = sc.nextLine();  
  
        System.out.println("Este nome tem " + nome.length() + " letras.");  
  
        System.out.println("Deseja continuar? (S/N)");  
        opcao = sc.nextLine();  
    } while (opcao.toUpperCase().equals("S"));  
}
```

Comandos **break** e **continue**

- O comando **break** é usado para forçar a saída de um bloco.
 - É utilizado com os comandos **switch**, **for**, **while** e **do...while**.
- O comando **continue** é usado para interromper uma iteração do laço de repetição.
 - Após seu uso, realiza novamente o teste da expressão booleana do laço de repetição.

Exemplo Break

- Lê números entre 1 e 10 informados pelo usuário. Aborta o programa caso números fora dessa faixa sejam informados.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    for (int i = 0; i < 5; i++) {  
        System.out.println("Informe um número entre 1 e 10: ");  
        int num = sc.nextInt();  
  
        if (num < 1 || num > 10) {  
            System.out.println("Número inválido! Abortando programa!");  
            break;  
        }  
    }  
}
```

Exemplo Continue

- Lê números entre 1 e 10 informados pelo usuário. Caso seja informado um número fora dessa faixa, avisa o usuário e solicita o número novamente.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    int cont = 0;  
  
    while (cont < 5) {  
        System.out.println("Informe um número entre 1 e 10: ");  
        int num = sc.nextInt();  
  
        if (num < 1 || num > 10) {  
            System.out.println("Número inválido! Informe novamente.");  
            continue;  
        }  
  
        cont++;  
    }  
}
```

Considerações Finais

- Nesta aula revisamos alguns conceitos.
- Métodos em Java seguindo a semântica de procedimentos e funções.
 - Vimos como modularizar um programa através de bibliotecas, ou arquivos de código fonte, separados do programa principal.
- O funcionamento dos laços de repetição em Java.
 - Os laços **while** e **for** realizam a iteração com uma condição booleana no início do bloco.
 - O laço **do...while** realiza a iteração com uma condição booleana no final do bloco.
 - Também revisamos os conceitos de **contadores** e **acumuladores**.
- Vimos como fazer a **depuração (debug)** de código Java.

Exercícios

- Utilizar a plataforma *beecrowd* para realização dos exercícios.
- Resolver os exercícios da lista:
 - Lista 2 - Comandos de repetição.