Programação Estruturada

Aula 08 - Funções e procedimentos

Apresentação

Na aula passada, você aprendeu em mais detalhes como funciona a manipulação de Strings. Já nesta aula, você irá aprender a criar suas próprias funções e procedimentos dentro do seu programa. O uso das funções e procedimentos no código é também chamada de modularização, significando a criação de pequenos módulos (funções ou procedimentos) que podem ser combinados para resolver problemas mais complexos.

Aprender a programar de forma modularizada implica na construção de programas mais legíveis e com um grande salto de produtividade, pois o lema de dividir um problema grande e complexo em problemas menores simplifica a tarefa do programador. Você verá que podemos dividir os programas em módulos e que essa divisão é uma das alternativas mais utilizadas para desenvolver grandes programas.

Faça uma boa leitura!

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Descrever e aplicar o conceito de modularização em programas de computador.
- Criar e usar funções nesses programas.
- Criar e usar procedimentos nesses programas.

1. Introdução

A melhor maneira de desenvolver um grande programa é construí-lo a partir de pequenas partes, as quais chamaremos de módulos. Essa técnica é chamada de “dividir para conquistar”, estratégia utilizada há muitos anos em ambientes de guerra. Uma grande vantagem dessa técnica é que a nossa atenção será focada em dividir um problema grande e complexo em menores e mais simples, e depois resolver de cada um deles mais facilmente. No final, ao resolvermos todos os problemas menores, combinamos as pequenas soluções em uma solução maior que resolve o problema originalmente grande e complexo. Percebeu qual será nossa estratégia?

Esse processo de modularização é feito geralmente através da criação e do uso de funções ou procedimentos. Esses dois conceitos formam o assunto desta aula. Eles trazem grandes vantagens para o programador, como você vai ver a partir de agora.

2. Conceito de módulos

Os módulos de um programa podem representar coisas distintas. Para esse curso, os módulos serão rotinas (funções ou procedimentos), as quais são consideradas as essências da programação estruturada. Tais módulos servem para dividir um grande programa em diversas partes menores, conforme explicamos na apresentação desse curso. Para seguir as regras de uma boa prática de programação, sempre que possível, se faz necessária a divisão do código, a fim de que não sejam construídos códigos extensos e de difícil entendimento e manutenção.

A modularização permite que cada módulo do programa seja escrito, testado e revisado individualmente, sem alterar o funcionamento do programa como um todo. E essa é uma grande vantagem quando desenvolvemos sistemas grandes e

complexos. Os programadores poderão desenvolver e testar os módulos separadamente, permitindo, inclusive, encontrar e solucionar erros sem que todos os módulos estejam completamente prontos.

A técnica da modularização também permite o trabalho em equipe, no qual vários desenvolvedores se dividem e cada um fica responsável por um módulo, ganhando-se, dessa forma, economia de tempo na construção dos programas.

Também é necessário compreendermos que a maioria das linguagens de programação disponibiliza um conjunto de módulos prontos aos programadores, úteis por reunirem de uma maneira organizada e concisa funções testadas de uso mais comum (como funções matemáticas, por exemplo). Esses módulos são as chamadas bibliotecas de funções e procedimentos. Você usou várias funções na aula de manipulação de Strings, certo?

A seguir, conheceremos os conceitos relacionados às funções e aos procedimentos.

3. Funções

Você sabe o que são funções? **Funções são estruturas de código que permitem o usuário organizar seus programas em partes menores e mais simples**. Se elas não existissem, os programas seriam mais complexos, com bastante duplicação de código, etc. De fato, isso ocorria nas primeiras linguagens de programação. Era uma tarefa difícil detectar e corrigir os erros lógicos e de sintaxe da linguagem. Para fazermos programas grandes e complexos, temos de construí-los em blocos, dessa forma, teremos um código de mais fácil compreensão e melhor organizado.

Mas, vamos ver como isso funciona na prática. Imagine um programa que calcula a média obtida por um aluno nas suas quatro notas de uma matéria. Esse é um problema pequeno e simples, vamos ver como você resolveria com os conhecimentos mostrados até agora:

```
import java.util.Scanner;

public class ProgramaMedia {
    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        System.out.println("Digite a primeira nota:");
        double nota1 = leitor.nextDouble();
        System.out.println("Digite a segunda nota:");
        double nota2 = leitor.nextDouble();
        System.out.println("Digite a terceira nota:");
        double nota3 = leitor.nextDouble();
        System.out.println("Digite a quarta nota:");
        double nota4 = leitor.nextDouble();
        double media = (nota1 + nota2 + nota3 + nota4)/4;
        System.out.println("Sua média foi: " + media);
    }
}
```

O ProgramaMedia lê quatro notas do teclado e calcula a média aritmética obtida com as quatro notas digitadas. Muito bem, imagine agora que em outras partes do seu programa você vai ter que fazer esse mesmo cálculo. Você teria que escrever novamente essa fórmula, certo? Vai escrever a fórmula uma vez em cada lugar que ela precisar ser utilizada. Isso é ruim, porque você estará duplicando código. Caso você precise alterar a fórmula, vai ter que alterar em vários lugares. Por exemplo, caso você queira mudar de média aritmética para média ponderada, vai ter que alterar a fórmula em vários lugares. Além do maior trabalho de procurar e trocar a fórmula diversas vezes, você corre o risco de esquecer-se de alterar uma delas e de o programa ficar inconsistente!

Para solucionar esse problema, você pode estruturar melhor o programa com o uso de uma função. Veja só como fica:

```
1  import java.util.Scanner;
2
3  public class ProgramaFuncaoMedia {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite a primeira nota:");
7          double nota1 = leitor.nextDouble();
8          System.out.println("Digite a segunda nota:");
9          double nota2 = leitor.nextDouble();
10         System.out.println("Digite a terceira nota:");
11         double nota3 = leitor.nextDouble();
12         System.out.println("Digite a quarta nota:");
13         double nota4 = leitor.nextDouble();
14         double media = calcularMedia(nota1, nota2, nota3, nota4);
15         System.out.println("Sua média foi: " + media);
16     }
17
18     public static double calcularMedia(double nota1, double nota2, double nota3, double nota4) {
19         return (nota1 + nota2 + nota3 + nota4)/4;
20     }
21 }
```

Note que no ProgramaFuncaoMedia existe a rotina main e agora uma rotina **calcularMedia()**. Essa rotina é chamada de função porque ela retorna um valor, a média calculada de acordo com as notas passadas como parâmetro. Ao definir a função calcularMedia(), você pode executá-la a partir da rotina main, como mostrado na linha 14 do programa.

Se você observar bem, em termos de número de linhas de código, o programa até aumentou nesse exemplo específico. Porém, agora você pode chamar a execução da função calcularMedia(), a partir de qualquer parte do seu programa. Isso faz com que você evite a duplicação do código que está dentro da função. Dizemos aqui que o código de calcular a média está encapsulado dentro da função, já que a rotina main agora só faz chamar a função, sem saber como ela está implementada. Isso permite que você altere o corpo da função sem precisar alterar o corpo da rotina main.

Entrando em mais detalhes, uma função em Java tem a seguinte forma geral:

```
1  public static tipo_de_retorno  <nome_da_função> (declaração_de_parâmetros)
2  {
3      corpo_da_função;
4  }
```

O tipo-de-retorno é o tipo do valor que a função vai retornar. Já o nome da função indica o nome pelo qual o bloco de código correspondente à função será chamado. É através do nome da função que acionamos sua execução a partir de outras partes do código, como a partir da rotina main. As regras para nomear uma função são as mesmas que utilizamos para nomear as nossas variáveis.

No corpo da função, podemos declarar variáveis (chamadas de variáveis locais) e comandos da linguagem. Uma variável local somente pode ter seu valor acessado dentro da função em que foi declarada. Um módulo pode também não ter nenhuma variável local. Por fim, a declaração de parâmetros é uma lista com a seguinte forma geral:

```
tipo nome1, tipo nome2, ... , tipo nomeN
```

Repare que o tipo deve ser especificado para cada uma das N variáveis de entrada. É na declaração de parâmetros que informamos ao compilador quais serão as entradas da função (assim como informamos a saída – tipo de retorno).

Vejamos um outro exemplo de código organizado em funções:

```
import java.util.Scanner;

public class ProgramaFuncaoQuadrado {
   public static void main(String[] args) {
      Scanner leitor = new Scanner(System.in);
      System.out.println("Digite um número:");
      double num = leitor.nextDouble();
      System.out.println("O quadrado desse número é:" + quadrado(num));
      // ... Aqui o programa faz um monte de coisa
      // ... Aqui o programa faz um monte de coisa
      // ... Aqui o programa faz um monte de coisa
      // ... Aqui o programa faz um monte de coisa
      // ... Depois de muito código, precisa-se calcular
      // ...    novamente o quadrado
      double num2 = leitor.nextDouble();
      System.out.println("O quadrado do segundo número digitado é:" + quadrado(num2));
   }

   public static double quadrado(double numero) {
      return numero*numero;
   }

}
```

Vamos fazer uma análise do ProgramaFuncaoQuadrado. Esse programa declara uma função chamada quadrado, que, dado um número, retorna o valor desse número multiplicado por ele mesmo, ou seja, esse número elevado ao quadrado. Essa função é então chamada em dois locais da rotina main. A função evita assim que a fórmula seja escrita em dois lugares diferentes. Isso é o que chamamos de promover o reuso de código. E apesar de estarmos trabalhando com funções bem simples por enquanto, essas funções poderiam ter, digamos, 200 linhas de código!

Vídeo 02 - Função Media

Como dito no início da aula, as funções vão permitir que você divida o problema em partes menores que depois possam ser combinadas para resolver um problema maior. Imagine então o caso de um programa que quer saber se um determinado aluno passou por média ou se está na final. Note que esse problema pode ser quebrado em dois. O primeiro trata de calcular a média do aluno. Já o segundo, de verificar se com essa média o aluno está aprovado ou não. Veja como podemos resolver esse problema fazendo o uso de funções:

```java
import java.util.Scanner;

public class ProgramaFuncaoMedia2 {
    public static final String APROVADO = "Aprovado";
    public static final String REPROVADO = "Reprovado";
    public static final String RECUPERACAO = "Em recuperação";

    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
            System.out.println("Digite a primeira nota:");
        double nota1 = leitor.nextDouble();
        System.out.println("Digite a segunda nota:");
        double nota2 = leitor.nextDouble();
        System.out.println("Digite a terceira nota:");
        double nota3 = leitor.nextDouble();
        System.out.println("Digite a quarta nota:");
        double nota4 = leitor.nextDouble();
        double media = calcularMedia(nota1, nota2, nota3, nota4);
        System.out.println("Sua média foi: " + media);
        System.out.println("Resultado, você está " + verificarSituacaoAluno(media));
    }

    public static String verificarSituacaoAluno(double media) {
        if (media >= 7) {
            return APROVADO;
        } else if (media < 3) {
            return REPROVADO;
        } else {
            return RECUPERACAO;
        }
    }

    public static double calcularMedia(double nota1, double nota2, double nota3, double nota4) {
        return (nota1 + nota2 + nota3 + nota4)/4;
    }
}
```

A primeira novidade que destacamos no ProgramaFuncaoMedia2 é a declaração de constantes que representam o texto que queremos retornar. Essa é uma boa prática de programação, porque se quisermos mudar o texto, basta alterar nas constantes. Além disso, se precisarmos comparar o resultado da função com os possíveis valores que ela pode retornar, fazemos uso das constantes.

Na rotina main, temos as chamadas para duas funções, a calcularMedia() e a **verificarSituacaoAluno**(). A primeira, nós já comentamos, é igual àquela declarada no ProgramaFuncaoMedia. Já a segunda, recebe uma média, a qual foi calculada

pela função calcularMedia() e retorna o texto correspondente ao status do aluno (aprovado, reprovado ou em recuperação).

Vídeo 03 - Função Acoplada

Vídeo 04 - Procedimentos

Note que com o uso das funções, o código da rotina main tende a ser pequeno, assim como o de todas as funções. Esse é um dos benefícios de usarmos funções. Nosso código vai ficando mais simples de ser entendido e alterado.

Atividade 01

1. Implemente e execute os ProgramaMedia e ProgramaFuncaoMedia como mostrado em aula. Compare as duas implementações, se resultam em dados diferentes, se tem vantagem usar uma ou outra implementação etc.
2. Implemente o ProgramaFuncaoQuadrado. Execute e veja seu funcionamento. Em seguida, altere fazendo com que o programa chame a função quadrado mais duas vezes dentro da rotina main. Comente sobre as possíveis vantagens de se ter estruturado o programa com funções.

4. Procedimentos

Vamos ao segundo tipo de rotina que podemos criar, os chamados procedimentos, que são as rotinas que não retornam valor. É isso mesmo, essa é a única diferença entre funções e procedimentos! Por isso, é comum algumas pessoas

no dia a dia se referirem a funções e procedimentos como uma coisa só. Evite isso! Use sempre a terminologia certa para cada tipo de rotina.

Você já deve ter percebido que já faz uso de um procedimento. Sabe de qual procedimento estou falando? Identifique-o no programa a seguir:

```
1  import java.util.Scanner;
2
3  public class ProgramaOla {
4      public static void main(String[] args) {
5          Scanner leitor = new Scanner(System.in);
6          System.out.println("Digite seu nome:");
7          String nome = leitor.nextLine();
8          System.out.println("Olá, " + nome);
9      }
10 }
```

Conseguiu identificar? É a rotina main. Ela é um procedimento! É uma rotina que recebe como argumento um array de Strings e que não retorna nada. Isso pode ser verificado pela palavra chave **void** que está no lugar do tipo de retorno. O void representa o conjunto vazio, indicando que não há valor válido a ser retornado. Em outras palavras, o void indica que a rotina não retorna nada, ou seja, que ela é um procedimento.

Dessa forma, o que muda na forma geral de se declarar um procedimento é a presença do void no lugar do tipo de retorno:

```
1  public static void <nome_do_procedimento>  (declaração_de_parâmetros)
2  {
3      corpo_da_função;
4  }
```

Vejamos um exemplo de código que faz uso de outros procedimentos além do main:

```java
import java.util.Scanner;

public class ProgramaFuncaoMedia3 {
    public static final String APROVADO = "Aprovado";
    public static final String REPROVADO = "Reprovado";
    public static final String RECUPERACAO = "Em recuperação";

    public static void main(String[] args) {
        Scanner leitor = new Scanner(System.in);
        pegarNotas(leitor);
    }

    private static void pegarNotas(Scanner leitor) {
        System.out.println("Digite a primeira nota:");
        double nota1 = leitor.nextDouble();
        System.out.println("Digite a segunda nota:");
        double nota2 = leitor.nextDouble();
        System.out.println("Digite a terceira nota:");
        double nota3 = leitor.nextDouble();
        System.out.println("Digite a quarta nota:");
        double nota4 = leitor.nextDouble();
        double media = calcularMedia(nota1, nota2, nota3, nota4);
        System.out.println("Sua média foi: " + media);
        System.out.println("Resultado, você está " + verificarSituacaoAluno(media));
    }

    public static String verificarSituacaoAluno(double media) {
        if (media >= 7) {
            return APROVADO;
        } else if (media < 3) {
            return REPROVADO;
        } else {
            return RECUPERACAO;
        }
    }

    public static double calcularMedia(double nota1, double nota2, double nota3, double nota4) {
        return (nota1 + nota2 + nota3 + nota4)/4;
    }
}
```

Essa nova versão do programa que calcula a média de 4 notas e da situação do aluno agora tem um procedimento chamado de pegarNotas. Esse procedimento está encapsulando o código que lê as notas, que chama as funções calcularMedia() e verificarSituacaoAluno(), e que imprime as informações na tela.

Vamos agora observar dois pontos importantes no código do ProgramaFuncaoMedia3. Em primeiro lugar, observe que uma rotina (função ou procedimento) pode chamar outra rotina. No caso, a rotina main está chamando a rotina pegarNotas() que por sua vez chama as rotinas calcularMedia() e verificarSituacaoAluno(). É muito comum em sistemas grandes observarmos essas situações de funções e procedimentos que chamam outras funções e procedimentos.

O segundo ponto que você pode inclusive estar se perguntando é se vale à pena criar o procedimento pegarNotas(). Essa questão é um pouco mais complicada. Olhando só o código mostrado, temos uma simplificação extrema da rotina main. Para algumas pessoas, isso pode não ser um benefício suficiente para justificar a criação de uma nova rotina. Porém, a criação da rotina pegarNotas() permite que outras partes do sistema (outras classes) façam uso dela. Se isso for necessário, passa a ser interessante a criação dessa rotina pegarNotas().

Atividade 02

1. Implemente e execute o ProgramaFuncaoMedia3, como mostrado em aula. Funcionando tudo corretamente, altere em seguida a rotina main para chamar novamente o procedimento pegarNotas(). Reporte o que mudou no comportamento do programa e qual foi o esforço necessário para fazer isso.

Conclusão

Terminamos assim o conteúdo teórico desta nossa aula. Para a próxima aula, continuaremos o assunto estudando sobre o uso de funções e procedimentos recursivos. Essa é uma abordagem que vai facilitar a construção de soluções para determinados tipos de problemas. Até lá!

5. Resumo

Nesta aula, você aprendeu conceitos relacionados à modularidade de programas de computador, em especial, no uso de funções e procedimentos. Esses dois tipos de rotinas que podemos criar nos permitem dividir problemas grandes e complexos em problemas menores e mais simples de se resolver. Uma vez resolvidos, esses pequenos módulos são combinados para solucionar o problema maior e mais complexo!

6. Autoavaliação

1. Qual o papel da modularização na criação de programas de computador?
2. O que significa o termo "dividir para conquistar"?
3. Qual a diferença entre funções e procedimentos?

7. Referências

ASCENCIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da programação de computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Editora Pearson, 2008.

THE JAVA tutorials. Disponível em: <http://download.oracle.com/javase/tutorial/>. Acesso em: 6 dez. 2011.