Técnicas de Programação

José Marcio Benite Ramos
Liluyoud Cury de Lacerda
Sara Luize Oliveira Duarte

INSTITUTO FEDERAL
RONDÔNIA

Cuiabá-MT
2014

Presidência da República Federativa do Brasil
Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
Diretoria de Integração das Redes de Educação Profissional e Tecnológica

Equipe de Revisão

Universidade Federal de Mato Grosso – UFMT

Coordenação Institucional
Carlos Rinaldi

Coordenação de Produção de Material Didático Impresso
Pedro Roberto Piloni

Designer Educacional
Claudinet Antônio Coltri Júnior

Designer Master
Neure Rejane Alves da Silva

Ilustração
Tatiane Hirata

Diagramação
Tatiane Hirata

Revisão de Língua Portuguesa
Ewerton Viegas Romeo Miranda

Revisão Final
Claudinet Antonio Coltri Junior

Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO

Campus Porto Velho Zona Norte

Direção-Geral
Miguel Fabrício Zamberlan

Direção de Administração e Planejamento
Gilberto Laske

Departamento de Produção de EaD
Ariádne Joseane Felix Quintela

Coordenação de Design Visual e Ambientes de Aprendizagem
Rafael Nink de Carvalho

Coordenação da Rede E-Tec
Ruth Aparecida Viana da Silva

Projeto Gráfico
Rede e-Tec Brasil / UFMT

Técnicas de Programação - Informática para Internet

R1756t Ramos, José Marcio Benite.

Técnicas de programação / Liluyoud Cury de Lacerda; Sara Luize Oliveira Duarte; org. Instituto Federal de Educação, Ciência e Tecnologia; Universidade Federal do Mato Grosso - Cuiabá : UFMT; Porto Velho: IFRO, 2014.

83 p. ; -- cm.
Curso Informática para internet.
ISBN 978-85-68172-04-9

1. Programação de computadores - Conceitos. 2. Banco de dados - Gerência . I. Lacerda,Liluyoud Cury de. II. Duarte, Sara Luize Oliveira. III. Instituto Federal de Educação, Ciência e Tecnologia. IV. Universidade Federal do Mato Grosso. V. Título.

CDD 005.7565
CDU 004.42

Ficha Catalográfica Elaborada pela Bibliotecária Cleuza Diogo Antunes CRB 11/864

Apresentação Rede e-Tec Brasil

Prezado(a) estudante,

Bem-vindo(a) à Rede e-Tec Brasil!

Você faz parte de uma rede nacional de ensino que, por sua vez, constitui uma das ações do Pronatec - Programa Nacional de Acesso ao Ensino Técnico e Emprego. O Pronatec, instituído pela Lei nº 12.513/2011, tem como objetivo principal expandir, interiorizar e democratizar a oferta de cursos de Educação Profissional e Tecnológica (EPT) para a população brasileira propiciando caminho de acesso mais rápido ao emprego.

É neste âmbito que as ações da Rede e-Tec Brasil promovem a parceria entre a Secretaria de Educação Profissional e Tecnológica (Setec) e as instâncias promotoras de ensino técnico, como os institutos federais, as secretarias de educação dos estados, as universidades, as escolas e colégios tecnológicos e o Sistema S.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade e ao promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

A Rede e-Tec Brasil leva diversos cursos técnicos a todas as regiões do país, incentivando os estudantes a concluir o ensino médio e a realizar uma formação e atualização contínuas. Os cursos são ofertados pelas instituições de educação profissional e o atendimento ao estudante é realizado tanto nas sedes das instituições quanto em suas unidades remotas, os polos.

Os parceiros da Rede e-Tec Brasil acreditam em uma educação profissional qualificada – integradora do ensino médio e da educação técnica – capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!
Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2014

Nosso contato
etecbrasil@mec.gov.br

Indicação de Ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: remete o tema para outras fontes: livros, filmes, músicas, *sites*, programas de TV.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Reflita: momento de uma pausa na leitura para refletir/escrever sobre pontos importantes e/ou questionamentos.

Palavra dos Professores-autores

Caro(a) estudante,

Seja bem-vindo(a) à disciplina de Técnicas de Programação.
Iniciaremos nossa jornada rumo ao conhecimento das técnicas de programação, em que veremos conceitos, boas práticas e mecanismos para criação de programas utilizando uma linguagem de computador.
Parabenizamos sua iniciativa de realizar este curso. Somos seres humanos, e como tais, temos um temor inicial ao adentrar em um mundo desconhecido. Mas somos movidos pela curiosidade, pela vontade de aprender mais, e essa força nos permite enfrentar esse receio e conquistar novos mundos.
O esforço empreendido durante esta jornada será recompensado com o conhecimento, o maior bem que podemos ter. O conhecimento adquirido nunca será tirado de você. Ele pode ser compartilhado, mas nunca subtraído, e é através dele que podemos abrir novas portas, aproveitar as oportunidades que o mercado irá nos oferecer.
Vale ressaltar que este material é apenas uma parte de um conjunto de acontecimentos que compõem a trajetória dos estudos que você deverá realizar. Deverá ser complementado através da utilização do ambiente virtual, do apoio do professor tutor, do aprofundamento dos conteúdos proporcionados pelos livros indicados etc. Este material é o início para o aprendizado do que a disciplina deve oferecer. Não deixe de acompanhar o recurso "saiba mais", que fará indicações de ampliações possíveis e importantes para o aprofundamento do aprendizado.
Muito importante também é a realização das atividades propostas. Nunca deixe de executá-las, é o momento indispensável de trabalho, no qual você irá imergir na disciplina com suas reflexões.

Bom estudo.
Professores.

Apresentação da Disciplina

A disciplina de Técnicas de Programação compreende os conceitos e boas práticas para a criação de programas de computadores, através de técnicas e recursos como: variáveis, tipos de dados, modularização, estruturas de controle, funções e procedimentos e recursividade.

Veremos que a construção de programas deve possuir características como correção, flexibilidade, eficiência e facilidade de compreensão.

O entendimento e utilização das variáveis, tipos de dados, modularização, estruturas de controle, funções, procedimentos e recursividade, são essenciais à formação do(a) profissional da área de tecnologias da informação.

Ao final desta disciplina você deverá possuir habilidades para desenvolver programas de computador, seguindo as especificações e paradigmas da lógica de programação e das linguagens de programação. Você será preparado(a) para utilizar as técnicas de refinamento sucessivo, recursividade e modularização, além dos conceitos de manipulação de dados na memória do computador. Com isso, será capaz de criar programas utilizando recursos de tipos de dados, funções e procedimentos, os quais possibilitarão mais eficiência e melhor desempenho dos programas criados.

É essencial que você relembre os conceitos utilizados em disciplinas anteriores para um melhor aproveitamento desta disciplina.

As aulas estão estruturadas de forma a conduzi-lo(a) às técnicas de programação e dar a visão para utilização dos recursos mais adequados para criação de programas de computadores.

Sumário

Aula 1. Conceitos básicos de programação

Objetivos:

- entender o que é um algoritmo;
- criar uma aplicação;
- reconhecer o conceito de compilação de aplicações; e
- conceituar entrada e saída de dados.

Olá! Seja bem-vindo(a) à nossa primeira aula da disciplina de Técnicas de Programação. Nesta aula veremos as ferramentas necessárias para criarmos um programa de computador e executá-lo. Inicialmente serão abordados os principais conceitos para compreendermos como um programa é organizado e executado pelo computador e como podemos interagir com ele, isto é, enviar e receber informações.

Vamos começar?

Bem, para iniciarmos a nossa aula vamos lembrar que podemos associar a cada tarefa que realizamos cotidianamente um conjunto de ações que devem ser executadas, muitas vezes em uma ordem predefinida, para que possamos concluí-las. Fazer um simples café ou trocar o pneu de um veículo são tarefas que necessitam de um planejamento de ações para que possamos, ao final deste, cumpri-las. Em computação, damos o nome de algoritmo a esse planejamento das ações, no qual definimos a ordem, a repetição e as condições que cada ação deverá ser executada.

Vamos ver o exemplo de uma receita de bolo:

Receita de bolo
Misture os ingredientes
Unte a forma com manteiga
Despeje a mistura na forma
Se houve coco ralado então despeje sobre a mistura
Leve a forma ao forno
Enquanto não corar deixe a forma no forno
Retire do forno
Deixe esfriar

Figura 01: esquema de um algoritmo em linguagem natural.
Fonte: Internet - http://www.vivaolinux.com.br/artigo/Otimizacao-de-algoritmos?pagina=2, acessado em 05/05/2013.

Agora que compreendeu o que é um algoritmo, podemos entender o que é um programa.

Os programas de computadores podem ser considerados como algoritmos escritos numa linguagem de computador (Java, C#, entre outras) e que por este são executados.

Devem ser definidas as instruções que o programa deverá executar, respeitando a ordem estabelecida, as condições necessárias e a necessidade de repetir determinado processo.

Podemos associar um programa de computador às instruções de um manual, em que se seguirmos as especificações, podemos montar um determinado produto ou até mesmo fazer um bolo.

Os programas de computador, escritos em uma linguagem de programação, devem ser "traduzidos" para a linguagem que a máquina possa compreender (Programa Objeto).

Figura 02: Conversão de Algoritmo em Programa Objeto.
Fonte: autor

1.1 Criando aplicações

O código fonte poderá ser criado em um editor de texto ASCII puro. Por exemplo: Bloco de Notas do Windows.

A-Z

Um editor de texto ASCII é aquele que só gera caracteres ASCII, isto é sem qualquer tipo de informações de formatação.

Depois de digitado, o mesmo deverá ser salvo com um nome e com a extensão ".java".

A seguir, veja o código fonte da aplicação "Olá Mundo!" (arquivo: OlaMundo.java):

```
public class OlaMundo {

    public static void main (String args[]) {

          System.out.println("Olá Mundo!");

    }

}
```

Obs.: o nome do arquivo deverá ter o mesmo nome do programa criado.

public class OlaMundo -> OlaMundo.Java

Após a criação do código, é preciso transformá-lo em uma linguagem mais próxima ao computador. É o que chamamos de compilação.

1.2 Compilando aplicações

Compilar um programa é converter o código fonte, escrito na linguagem de programação, em um código objeto equivalente, o qual pode ser executado pelo computador.

A compilação de aplicações pode ser compreendida como a transformação do código fonte de uma linguagem de programação de alto nível (linguagem mais próxima da linguagem do programador, por exemplo, JAVA) em uma linguagem de programação em baixo nível (linguagem mais próxima do computador, por exemplo, *Assembly* ou Bytecode).

A-Z

Bytecode pode ser considerado como um estágio intermediário entre o código fonte (escrito numa linguagem de programação) e o programa final – executável. Ele é resultante de um processo semelhante à compilação. O bytecode será interpretado por um aplicativo que fará a sua execução.

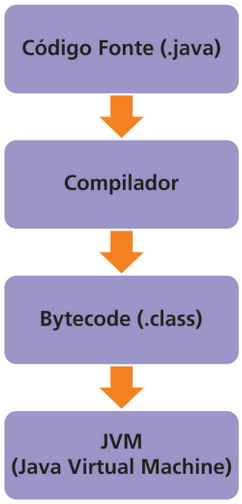


Figura 03: Compilação de um programa em JAVA.
Fonte: Internet - http://bug-digital.blogspot.com.br/2011/04/java-o-que-e-como-funciona.html, acessado em 05/05/2013.

Para compilar a aplicação, basta digitar o comando:

javac OlaMundo.java

Esse comando gera o arquivo OlaMundo.class, que é o bytecode da aplicação.

Para executá-lo basta digitar o comando:

java HelloWorld

Podemos também utilizar um IDE para escrever nossos programas e executá-los.

A-Z

IDE – Integrated Development Enviroment – é um ambiente integrado para desenvolvimento de software

1.2.1 Descrição dos comandos

Para que você compreenda melhor, a seguir faremos uma breve descrição dos comandos utilizados no exemplo anterior.

Acesse o *site* para maiores informações sobre o IDE Eclipse: **http://www.cin.ufpe.br/~phmb/ip/MaterialDeEnsino/IntroducaoAoEclipse/IntroducaoAoEclipse.html**

a) comentários:

// Comentário de uma linha, todo o restante da linha é ignorado.

/* Insira aqui o texto a ser ignorado */

b) class: palavra reservada que marca o início da declaração de uma classe. Os conceitos de classes serão apresentados na disciplina de Programação Orientada a Objetos.

c) public: define a visibilidade. Ver item "class".

d) OlaMundo: nome dado a classe.

e) static: indica que o método pode ser invocado, mesmo quando não foi criado o objeto para a classe. Ver item "class".

f) void: valor de retorno do método. Quando não retorna nenhum valor, ele retorna void (espécie de valor vazio que deve ser especificado).

g) main: nome particular de método que indica para o compilador o início do programa. Veremos como utilizar métodos na aula referente a funções e procedimentos.

h) (String args[ ]): argumento de main, um vetor de Strings formado quando são passados ou não argumentos através da invocação do programa. Esses comandos contêm conceitos, os quais serão vistos mais adiante. Sendo assim, vamos por enquanto apenas utilizá-los.

i) { ... }: delimitam um bloco de código. Definem também o início e fim da classe.

j) System.out.println("Olá Mundo!"): por hora, guardar essa linha de código como o comando para imprimir mensagens na tela.

k) ;: "ponto e vírgula" separa operações.

1.3 Operações de entrada e saída

Os cálculos realizados pelo computador não teriam nenhuma utilidade se não pudéssemos fornecer os dados sobre os quais esses cálculos serão efetuados e, segundo, ver os seus resultados.

Os programas de computadores devem receber os dados de fontes externas (entrada de dados, por exemplo, informados pelo usuário), efetuar os cálculos e fornecer uma saída, isto é, os resultados obtidos.

a) Saída: fornece os resultados obtidos. A saída para a tela (console da aplicação) usa a seguinte estrutura:

```
System.out.println("Imprime algo na tela");
```

A-Z

A classe scanner implementa as operações de entrada e saída de dados pelo teclado no console.

b) Entrada: operação de recepção dos dados. Utilizaremos a classe "Scanner" para a leitura dos dados informados pelo usuário e os armazenaremos em uma variável.

Para aprender mais sobre a Classe Scanner, acesse o *site*: **http://www.devmedia.com.br/entrada-de-dados-classe-scanner/21366**

Exemplo:

```
import java.util.Scanner;
...
//Declarando uma variável (objeto) do tipo Scanner
Scanner entrada = new Scanner(System.in);
//Declaração de variáveis serão vistas na aula seguinte.
Por enquanto vamos
// usá-la para podermos escrever o programa.
int valor;
System.out.println("informe o valor:");
//Utilizando a variável "entrada", da classe Scanner, para
obter a informação
//digitada pelo usuário. "entrada.nextInt()" irá ler um
valor inteiro informado na
// console
int dado = entrada.nextInt();
System.out.println("O valor informado foi: " + dado);
```

A primeira linha do exemplo define que iremos utilizar os recursos disponibilizados pela classe "Scanner", presente na biblioteca "java.util".

A classe "Scanner" tem como uma das vantagens a implementação de operações de entrada e saída de dados pelo teclado na console da aplicação.

Podemos obter outros tipos de valores da console através dos seguintes comandos:

```
System.out.print("digite uma linha: ");
// Ler uma linha de caracteres
String linha = s.nextLine();
System.out.print("digite um numero: ");
//Ler um número inteiro
int i = s.nextInt();
System.out.print("digite um numero: ");
//Ler um número decimal
double d = s.nextDouble();
```

Resumo

O sucesso da execução de uma tarefa depende da forma como vamos executá-la. Se planejarmos previamente as ações que serão realizadas, teremos maior chance de concluirmos a tarefa de forma mais eficiente. Vimos nessa aula que os computadores necessitam de instruções adequadas para resolver nossos problemas. Essas instruções devem ser escritas em uma linguagem que possa ser convertida de maneira que os computadores passem a entender nossas instruções. Os programas devem ser escritos e posteriormente compilados, processo necessário para podermos executá-los. As informações que passamos ao programa são chamadas de "Entradas" e as informações que o programa nos fornece são chamadas de "Saídas".

Atividades de aprendizagem

1. Dado o algoritmo abaixo, "Calcular Valor Pedido", identifique os dados de entrada, qual o processamento e os dados de saída:

- Leia o código do item.
- Leia o valor do item.
- Leia a quantidade do item.
- Calcule o valor total do pedido (quantidade * valor do item).
- Escreva o valor total do pedido.

2. Crie um programa que leia o nome do usuário e sua idade, depois escreva os valores obtidos.

3. Faça um programa que leia dois valores inteiros e armazene cada um em uma variável, depois troque os valores das duas variáveis entre si. Como sa-

ída, escreva os valores lidos.

E assim terminamos a nossa primeira aula da disciplina de Técnicas de Programação. Espero você na próxima aula. Até lá.

Aula 2. Variáveis e tipos de dados

Objetivos:

- conceituar tipos de dados;
- identificar o conceito de variáveis;
- reconhecer o que são "constantes";
- atribuir valores variáveis e constantes; e
- definir o que são tipos de dados primitivos.

Seja bem-vindo(a) à nossa segunda aula. Agora que você já compreendeu os conceitos iniciais da nossa disciplina, vamos aprofundar um pouquinho mais.

Pois bem, sabemos que na Matemática uma variável é a representação simbólica dos elementos de certo conjunto de valores. As variáveis são utilizadas para representar e armazenar um determinado valor em uma fórmula matemática, o qual não é conhecido ou pode variar.

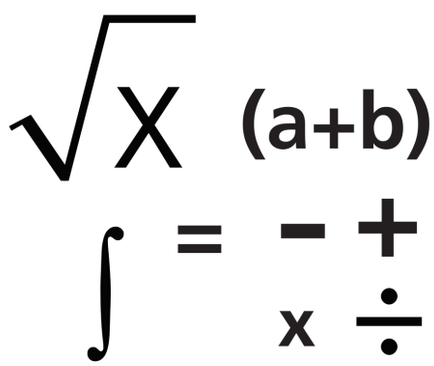


Figura 04: Variáveis e fórmulas matemáticas.
Fonte: Internet - http://mauricioromao.blog.br/vagas-legislativas-e-o-emprego-da-formula-dhondt-no-brasil/, acessado em 05/05/2013.

Podemos fazer uma comparação com algoritmos, os quais são destinados a resolver um problema no computador. Cada variável corresponde a uma posição de memória, cujo conteúdo pode variar ao longo do tempo durante a execução deste algoritmo. Embora a variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

A-Z

Programa de computador pode ser considerado como um conjunto de instruções (comandos) que descrevem uma tarefa a ser realizada pelo computador.

Nesta aula você verá que os computadores devem ser preparados para receberem as informações que serão fornecidas, apresentadas e tratadas pelos programas, através da utilização de variáveis. Os programas de computadores que serão desenvolvidos devem estabelecer um contrato com o sistema operacional do computador. As informações que serão tratadas devem respeitar os tipos definidos nesse contrato, isto é, quando você definir que irá trabalhar com números em um determinado momento, apenas valores numéricos (existem vários tipos numéricos) deverão ser informados.

Para tanto, você será apresentado(a) aos tipos mais comuns de dados suportados pela linguagem de programação (será utilizado o JAVA), como caracteres, números, datas, valores booleanos, e verá também como as informações são representadas em um programa e armazenadas na memória do computador através de variáveis.

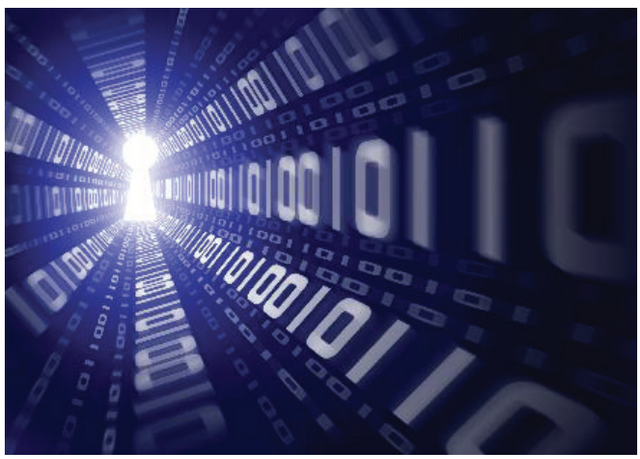


Figura 05: Bits formando uma luz no fim do túnel.
Fonte: Internet - http://scottkantner.com/bits-bytes-and-bandwidth/, acessado em 07/03/2013.

Vamos iniciar a nossa aula compreendendo o que são tipos de dados.

2.1 Tipos de dados

Os tipos de dados definem uma combinação de valores que uma variável pode receber e o conjunto de operações que esta pode executar.

Os valores expressos podem ser de forma genérica, numéricos (“1”, “25”, “5,3”), caracteres (“a”, “abc”, “João”), datas (“10/10/2012”, “20 de maio de 2010”) etc.

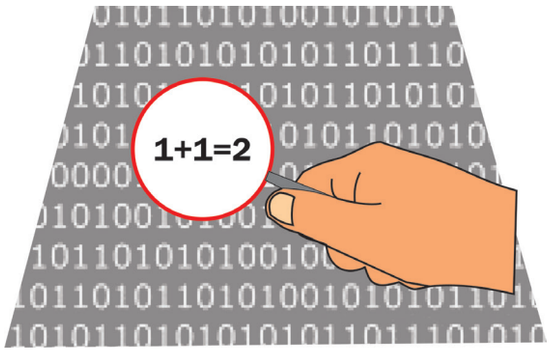


Figura 06: Tipo de dado.
Fonte: Internet - http://www.efetividade.blog.br/2011/05/26/voce-sabe-como-funciona-as-memorias-do-seu-pc/, acessado em: 07/03/2013.

Precisamos lembrar que as operações realizadas entre números do tipo in teiro são diferentes dos números reais (com casas decimais), os quais podem ser divididos.

O mesmo vale para tipos como caracteres que não possuem operações matemáticas, mas podem, por exemplo, ser concatenados (“ca” + “sa” = “casa”). Como as variáveis dependem do sistema operacional e da linguagem utilizada, elas podem possuir muitas variantes.

A-Z

Concatenar consiste em unir conjuntos de caracteres transformando-os em um só.

Até aqui está tudo bem? Conseguiu compreender o conceito? Vamos prosseguir?

Então vamos lá.

É importante salientar que definir o tipo de dado da informação que será manipulada é muito importante para o SO (Sistema Operacional) determinar o espaço que será reservado, pois é através dele que o SO solicitará uma porção da memória para armazenar a informação tratada.

A-Z

Um compilador é um programa que converte um código descrito em uma linguagem de alto nível para um programa em linguagem simbólica “compreensível” pelo processador.

Outra questão é que, a obtenção da informação da memória também depende do tipo de dado, pois são eles que possibilitam ao compilador realizar

as conversões dos dados em memória para obter os valores manipulados nos programas. Lembre-se, como foi apresentado em Fundamentos da Informática, os dados em memória são representados apenas por "0 e 1", e é através do tipo de dado que esse conjunto de números binários são definidos como uma letra ou um número.

Para pensar

Imagine que lhe peçamos para buscar um recipiente para armazenar uma "coisa". Sem saber o que é, você poderia pegar uma garrafa de 2 litros ou um pequeno saco de papel. Se pretendíamos lhe entregar um litro de água, a garrafa atenderia, mas o saco de papel não iria servir. Agora imagine que a quantidade poderá ser imensa, ficaria impossível idealizar o recipiente correto. Isso vale para o SO, quando um sistema vai manipular informações. Elas devem ser tipificadas para que não ocorra uma reserva de espaço em memória insuficiente ou extremamente grande.

Agora que já compreendemos a importância da definição do tipo de dados, vamos falar sobre as variáveis.

2.2 Variáveis

As variáveis são utilizadas pelos programas para armazenarem as informações manipuladas durante suas execuções. Elas representam espaços na memória RAM e, através delas, os programas podem acessá-las e/ou modificá-las.

A-Z

Hexadecimal é um sistema de numeração que representa os números em base 16 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

A memória principal (RAM) de um computador é dividida em pequenos pedaços, e cada um é identificado por um endereço (valor em hexadecimal). Todas as informações que serão manipuladas pelo programa devem ser armazenadas na memória.

Quando os programas necessitam usar as informações, eles devem se referenciar ao endereço onde elas estão armazenadas. Atualmente, as memórias podem possuir bilhões de endereços, todos representados em hexadecimal. Trabalhar com valores hexadecimais é muito complexo, assim usamos um apelido (nome) para uma posição na memória, a **variável**.

Apresentaremos a seguir algumas características que devem ser observadas ao criarmos uma variável:

- **Nome**: conjunto de caracteres que identificará a variável. Deve ser único dentro de seu escopo (espaço de atuação). Ex.: nome, data nascimento.

- **Valor**: (**conteúdo**) dado contido na variável.

- **Tipo**: tipo de dados que será armazenado nas variáveis. Como visto anteriormente, podem ser números: "12", "2,5"; caracteres: "olá", "abcd"; datas "12/12/2012", entre outros, que serão apresentados mais a frente.

A-Z

O escopo de uma variável é a parte do programa em que ela pode ser utilizada. O escopo pode ser também aplicado aos métodos.

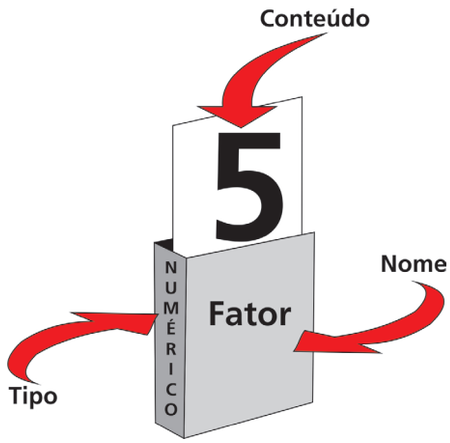


Figura 07: Estrutura de uma variável
Fonte: Internet - http://www.criarprogramas.com/2011/06/c-tipos-de-dados-e-declaracao-de-variaveis/, acessado em 07/03/2012.

Agora que já compreendemos as variáveis, vamos entender o conceito de "constantes".

2.3 Constantes

Constantes são endereços de memória destinados a armazenar informações fixas, inalteráveis, durante a execução do programa.

As constantes diferem das variáveis, pois estas podem ter seus valores alterados, já as constantes, como o próprio nome diz, não.

Toda constante representa um dado, portanto é necessário declarar seu tipo.

Para declararmos uma constante, utilizamos a palavra chave "final". Uma constante global (declarada no corpo da classe) deve ser precedida da palavra chave "static".

Representação:

```
static final int op1;
final int op2;
```

2.4 Declaração de variáveis

Como vimos anteriormente, variáveis correspondem a uma posição de memória, cujo conteúdo pode variar durante a execução do programa.

Embora possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

Toda variável é identificada por um nome ou identificador. Vamos ver como são as regras para a sua construção:

- Não usar palavras reservadas (comandos da linguagem).
- Possuir como 1° caractere uma letra ou "_".
- Ter como demais caracteres letras, números ou "_".
- Ter no máximo 127 caracteres.
- Não possuir espaços em branco.

Exemplos:

```
public float raio
public float comprimento
public int nota
public String nome
```

2.5 Atribuição de valores

A operação de atribuição consiste em fazer com que uma variável "receba" um determinado valor. Para atribuir valores a variáveis, devemos usar o sinal de "=".

```
a = 2;
b = 3;
c = a + b;
```

Exemplos:

Como vimos anteriormente, as constantes têm valores eternamente iguais. Para atribuição usamos o sinal de "=".

Exemplos:

PI = 3.14;

Variáveis podem ser atribuídas em forma de expressões, como:

```
int x, y, z;
x = y = z = 0;
```

As três variáveis x, y e z recebem o valor 0;

A seguir, apresentaremos um exemplo da utilização de variáveis e constantes:

```
public class Teste {
      static final float pi = 3.14;
      public static void main (String args[]) {
            float raio = 10;
            float comprimento = raio * 2 * pi;
            System.out.println("O comprimento é: " + com-
primento);
      }
}
```

2.6 Tipos de dados primitivos

Vimos que os **tipos de dados** são uma combinação de valores que uma variável pode armazenar, o que pode variar conforme o sistema operacional e a linguagem de programação. Assim, uma variável nomeada "nome" utilizada para armazenar nomes de pessoas deverá estar associada a um tipo de dado compatível com os valores armazenados.

De acordo com a linguagem de programação utilizada, o tipo de um dado é verificado de maneira diferente, conforme a análise do compilador. O Java é uma linguagem compilada e possui seu conjunto de tipos de dado básico.

Acesse o *site* para maiores informações sobre a linguagem JAVA: **http://www.java.com/pt_BR/download/faq/whatis_java.**

Esses tipos são conhecidos como tipos primitivos, pois são suportados diretamente pelo compilador, e são utilizados durante a codificação na definição de variáveis.

Tipo	Descrição	Tamanho
boolean	Pode assumir o valor true (verdadeiro) ou o valor false (falso)	1 bit
Char	Caractere em notação Unicode. Armazena dados alfanuméricos.	16 bits
Byte	Inteiro. Pode assumir valores entre -128 e 127.	8 bits
Short	Inteiro. Valores possíveis: -32.768 a 32.767	16 bits
Int	Inteiro. Valores entre -2.147.483.648 e 2.147.483.647.	32 bits
Long	Inteiro. Valores entre -9.223.372.036.854.770.000 e 9.223.372.036.854.770.000.	64 bits
Float	Real (representa números em notação de ponto flutuante). Valores entre -1,4024E-37 e 3.40282347E + 38	32 bits
double	Real (representa números em notação de ponto flutuante). Valores entre-4,94E-307 e 1.7976931348623157OE + 308	64 bits

A-Z

O UNICODE é uma notação que utiliza uma tabela contendo representação numérica dos caracteres.

Ponto flutuante é um formato de representação dos números reais, onde temos uma parte inteira a vírgula e a parte fracionária.

A tabela acima representa alguns dos principais tipos de dados utilizados para manipulação de informações em programas desenvolvidos na linguagem JAVA. A coluna "Tamanho" especifica o espaço de memória utilizado por cada tipo. Sendo assim, é muito importante especificar adequadamente o tipo empregado para não reservarmos espaços de memória que não serão utilizados e, também, não subdimensionar o espaço a ser utilizado.

Resumo

Vimos nessa aula que existem vários tipos de dados, os quais são utilizados para especificar que tipo de informação será armazenado em uma variável. A manipulação de informações depende das especificações corretas dos tipos de dados. O tipo de dado informa ao compilador que tipo de informação está sendo tratada, isto é, se os valores binários armazenados representam números, caracteres ou outros valores e permite que uma porção correta da memória seja reservada para o armazenamento da informação. Vimos também como manipulamos as variáveis, como atribuímos valores a uma variável e como podemos recuperá-los.

Atividades de aprendizagem

1. Informe qual o valor da variável resultado, em cada linha, após a execução das seguintes operações:

a) resultado = 3 + 6; x = 2; y = 3;

b) resultado = resultado / y – x;

c) resultado = 4; x = 2;

d) resultado = resultado * x;

2. Faça um programa que calcule a média de três valores informados pelo usuário.

3. Faça um programa que, dado uma temperatura em graus Celsius, converta-a para Fahrenheit, conforme a fórmula:

°F = (1.8 * °C) + 32

E então, fez todos os exercícios propostos? Consegui resolvê-los? Espero que sim. Assim terminamos o nosso segundo encontro. Espero por você na nossa terceira aula. Até lá!

Aula 3. Estruturas de controle

Objetivos:

- identificar os comandos de decisão;
- elaborar um comando de decisão;
- identificar os comandos de repetição; e
- elaborar um comando de repetição.

Olá! Estamos chegando à metade da nossa disciplina! Nesta aula, você será apresentado(a) às várias estruturas da linguagem utilizadas para mudar o fluxo sequencial de execução das instruções nos programas. Serão apresentadas estruturas de desvios (condicionais) que quebrarão o fluxo das instruções, permitindo que o programa siga por um caminho ou outro e estruturas de repetição que possibilitarão que um conjunto de instruções possa ser executado repetidas vezes.

Muitas vezes nos deparamos com situações em que devemos escolher um caminho, entre vários, para podermos prosseguir. Se estivermos diante de uma bifurcação, em nosso caminho, devemos fazer uma escolha entre seguir por um ou outro lado. Normalmente ponderamos os prós e contras para essa escolha e seguimos em frente. As execuções dos programas de computador também necessitam realizar escolhas para a correta resolução dos problemas, nesse caso os caminhos seriam as sequências de ações que devem ser executadas.

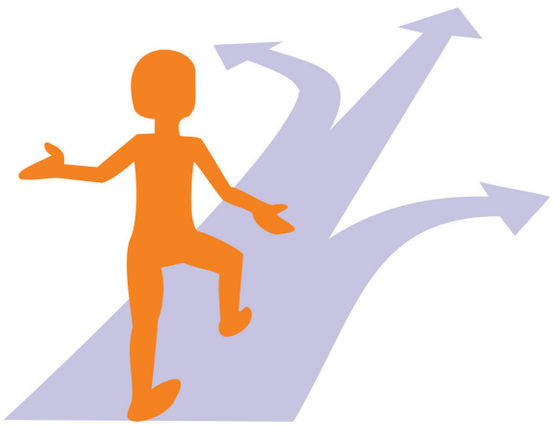


Figura 08: Escolha de um caminho
Fonte: Internet - http://www.vivaazul.com.br/artigo_op.php, acessado em 30/04/2013.

Os programas de computadores são basicamente compostos por uma sequência de comandos (instruções) organizados de tal maneira a resolver um problema proposto.

Essa forma de execução, em sequência, é denominada fluxo sequencial de execução.

Em muitos casos, são necessárias alternativas no fluxo de execução (bifurcação-seleção) ou a repetição de instruções (comandos) para a resolução de um problema.

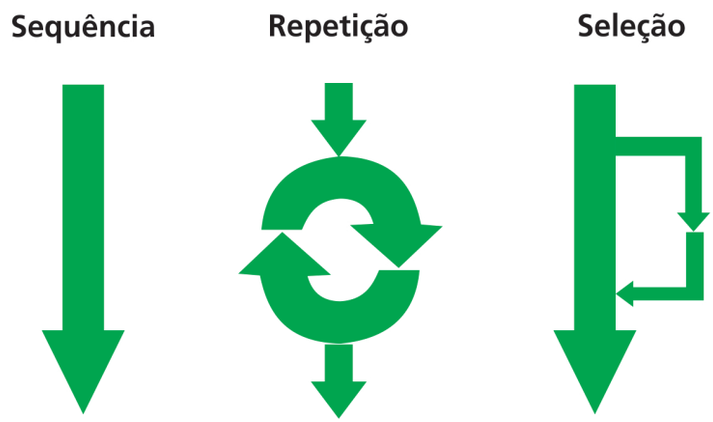


Figura 09: Fluxo de execução de instruções
Fonte: O autor.

Para que tudo isso aconteça, é necessária a tomada de decisão.

3.1 Comandos de decisão

Como o próprio nome indica, são utilizados para adotar uma decisão baseada no resultado da verificação de uma condição e seleciona um caminho a ser seguido, contendo as ações possíveis para serem executadas.

Os comandos de decisões fornecem desvios, possibilitando que o programa siga um caminho ou outro, isto é, proceda de uma ou outra maneira (executa ações em um caminho e outras ações no caminho alternativo). A seguir apresentaremos as principais estruturas de decisão:

- "Se Então";
- "Se Então Senão";

- "Ninhos de Se"; e
- "Caso".

Vamos aprender um pouco mais sobre cada um desses comandos.

3.1.1 Comando SE

Estrutura de decisão mais comum nos programas. Este comando realiza o desvio da sequência de instruções, permite a execução de uma instrução, ou bloco de instruções, somente se a condição de verificação, atribuída ao comando, for verdadeira. Em caso contrário, condição falsa, as instruções não serão executadas.

Sintaxe:

```
if (<condição>) comando;
```

Exemplo:

```
if (x > 0)
System.out.println("X é maior que zero.");
```

No exemplo acima, a saída para a console (comando System.out.println) será escrita apenas se o valor de x for maior que zero.

3.1.2 Comando SE - ENTÃO - SENÃO

O comando Se poderá ser acompanhado da cláusula SENÃO. Nesse caso, é criada uma bifurcação no fluxo de execução das instruções.

Utilizado para executar uma ação caso a condição seja verdadeira ou outra (SENÃO) caso seja falsa.

Sintaxe:

```
if (<condição>) comando;
else comando;
```

Exemplos:

```
if (x > 0)
System.out.println("X é maior que zero.");
else
System.out.println("X é menor que zero.");
```

No exemplo acima, a saída para a console ("X é maior que zero.") será escrita se o valor de x for maior que zero e a saída para a console ("X é menor que zero.") será escrita se o valor de x for menor que zero.

```
import java.util.Scanner;
…
public static void main(String args[]) throws IOEx-
ception {
Scanner entrada = new Scanner(System.in);
String nome;
double nota1, nota2, media;
System.out.println("Entre com o nome do aluno:");
nome = entrada.nextLine ();
System.out.println("Entre com a primeira nota:");
nota1 = entrada.nextDouble();
System.out.println("Entre com a segunda nota:");
nota2 = entrada.nextDouble ();
media = (nota1 + nota2) / 2;
if (media >= 6)
      System.out.println("Aprovado");
else
      System.out.println("Reprovado");
System.out.println(nome + " " + Double.
toString(media));
}
```

No exemplo acima, será escrito na console "Aprovado", se a média for maior ou igual a seis, e "Reprovado", se a média forma menor que seis. Os valores do nome e da média sempre serão escritos na console.

3.1.3 Comando ninhos de SE

Este comando é utilizado em ocasiões que necessitam a tomada de decisão para mais de duas opções.

Sintaxe:

```
if (<condição>) comando;
else if (<condição>) comando;
...
else comando;
```

Exemplos:

```
if (x == 0)
System.out.println(“X é zero.”);
else if (x == 1)
System.out.println(“X é um.”);
else
System.out.println(“X não é zero nem um.”);
```

No exemplo acima, a saída para a console (“X é zero.”) será escrita se o valor de x for igual a zero, a saída para a console (“X é um.”) será escrita se o valor de x for igual a um e a saída para a console (“X não é zero nem um.”) será escrita para os demais valores de x.

No exemplo abaixo, serão solicitados o nome e as duas notas de um aluno. Será verificada a sua condição: média maior ou igual a sete, aprovado; média entre três e sete, recuperação e média menor que três, reprovado.

```
import java.util.Scanner;
…
public static void main(String args[]) throws IOEx-
ception {
Scanner entrada = new Scanner(System.in);
String nome;
double nota1, nota2, media;
System.out.println("Entre com o nome do aluno:");
nome = entrada.nextLine ();
System.out.println("Entre com a primeira nota:");
nota1 = entrada.nextDouble();
System.out.println("Entre com a segunda nota:");
nota2 = entrada.nextDouble ();
media = (nota1 + nota2) / 2;
if (media >= 7)
      System.out.println("Aprovado");
else if ((media < 7) & (media >= 3))
      System.out.println("Prova Final");
else
      System.out.println("Reprovado");
System.out.println(nome + " " + Double.
toString(media));
}
```

3.1.4 Comando Caso

Podemos considerar o comando "Caso" como uma especialização do comando "Ninhos de SE", em que várias alternativas são apresentadas, sendo executados os comandos associados à alternativa que satisfaz a condição de verificação.

O comando Caso é utilizado para testar uma expressão ou o valor de uma variável (condição). Compara-se o resultado obtido no teste com os valores das cláusulas "Caso" (opções).

Sintaxe:

```
Switch VAR {
      case 1: comandos; break;
      case 2: comandos; break;
      default: comandos; break;
}
```

Obs.: nesse caso, VAR deverá ser do tipo "int";

default: será assumido caso o valor de VAR não seja igual a nenhuma das opções;

break: faz com que a estrutura seja interrompida após a execução do comando ou bloco de comandos.

Exemplo: informar um valor numérico referente a um mês e escrever o respectivo nome do mês.

```
import java.util.Scanner;
…
public static void main(String args[]) throws IOEx-
ception {
Scanner entrada = new Scanner(System.in);
String mes;
System.out.println("Entre com o valor do mês:");
mes = entrada.nexInt();
switch(monthNow){
      case 1: mes="Janeiro"; break;
      case 2: mes="Fevereiro"; break;
      case 3: mes="Março"; break;
      case 4: mes="Abril"; break;
      case 5: mes="Maio"; break;
      case 6: mes="Junho"; break;
      case 7: mes="Julho"; break;
      case 8: mes="Agosto"; break;
      case 9: mes="Setembro"; break;
      case 10: mes="Outubro"; break;
      case 11: mes="Novembro"; break;
      case 12: mes="Dezembro"; break;
      default: mes="Inválido"; break;
}
System.out.println("O mês informado foi: " + mes);
}
```

No exemplo acima, a saída para a console corresponderá à descrição do número associado ao mês informado. Se nenhum dos valores informados corresponderem a uma das opções do comando "CASO", o valor atribuído a variável "mes" será "Inválido".

3.2 Comandos de repetição

Em nosso cotidiano é comum nos depararmos com a repetição de procedimentos para realização de uma tarefa. A repetição não é infinita, isto é, em algum momento ela é interrompida, normalmente quando atingimos o objetivo.

Os comandos de repetição são utilizados para repetir a execução de um conjunto de comandos (instruções).

Serão apresentados três tipos de estrutura de repetição:

a) ENQUANTO: com teste no início;

while (**condição**) <comandos>

b) REPITA: com teste no final;

do <comandos> while (**condição**)

c) PARA: verificação automática.

for (c0, **condição**, c1) <comandos>

Como podemos observar nas estruturas dos comandos de repetição, todos têm em comum uma característica: a verificação da condição de parada. É através dela que determinamos um ponto de parada, o que determina o término da repetição e impede que o ciclo se torne infinito. Enquanto o resultado da verificação da condição retorna verdadeiro, a repetição continua.

A condição de parada deve possuir uma variável, para comparação, que deverá ser alterada durante os ciclos de repetição do comando e deve-se garantir que em algum momento o seu valor deixe de satisfazer a condição (passa a ser falsa), encerrando a repetição. Isso garante que os comandos de repetição entrem em "*looping*" infinito, isto é, fiquem repetindo indefinitivamente.

3.2.1 Comando Enquanto

Este comando tem como característica principal a verificação da condição de parada no início, isso possibilita a saída mesmo antes de iniciar a repetição.

O comando "ENQUANTO" permite a execução de um comando, ou conjunto de comandos, repetidamente, enquanto uma condição for verdadeira. Quando a condição não for mais satisfeita, ou seja, deixar de ser verdadeira, a repetição é interrompida.

Nesse caso, não é necessário conhecer o número de repetições.

Funcionamento:

1. Testa a condição de parada.

2. Se a condição for verdadeira, então executa o bloco dos comandos do REPITA.

3. Se a condição for falsa, sai da repetição. Executa o próximo comando após

o REPITA.

4. Após executar o último comando do bloco, volta ao passo 1.

Sintaxe:

```
while (condição) {
      <comando>
}
```

Exemplo: realizar a somatória dos salários informados pelo usuário. A repetição será interrompida quando o valor informado para o salário for menor que zero.

```
import java.util.Scanner;
…
public static void main(String args[]) throws IOEx-
ception {
Scanner entrada = new Scanner(System.in);
String nome;
double salario, total;
System.out.println("Entre com o nome:");
nome = entrada.nextLine();
salario = total = 0;
while (salario >= 0) {
      total = total + salario;
      System.out.println("Entre com o salario:");
      salario = entrada.nextDouble();
}
System.out.println("Nome: " + nome);
System.out.println("Total: " + total);
}
```

Obs.: de acordo com exemplo acima, podemos verificar que a repetição termina quando um valor negativo é atribuído para o salário.

3.2.2 Comando Repita

O comando repita é semelhante ao comando "ENQUANTO".

Ele comporta a execução de comandos repetidamente até que uma condição de controle seja FALSA. Entretanto, sua validação da condição é realizada no final, fazendo com que a repetição seja executada pelo menos uma vez.

Funcionamento:

1. Execução dos comandos dentro do bloco REPITA.

2. Testa a condição de parada.

3. Se a condição for verdadeira, volta ao passo 1.

4. Se a condição for falsa, então sai da repetição. Executa o próximo comando após o REPITA.

Sintaxe:

```
do {
<comando>
}
while (condição)
```

Exemplo: escrever os primeiros 100 números pares, considerando o número zero como o primeiro par.

```
public static void main(String args[]) throws IOEx-
ception {
int par, cont;
par = cont = 0;
do {
      System.out.print(par + ", ");
      par = par + 2;
      cont++;
}
while (cont < 100);
}
```

3.2.3 Comando Para

Assim como outros comandos de repetição, o comando PARA permite que um bloco de comando seja repetido um número específico de vezes.

O comando PARA repete o bloco de comandos enquanto a condição se mantiver verdadeira.

Este comando utiliza um contador (variável que é incrementada a cada laço da repetição) em sua estrutura usada para testar a condição. O contador é inicializado no início da execução do comando. Em seguida, a condição é tes-

tada, e caso seja verdadeira, os comandos são executados. Após a execução dos comandos e antes de verificar a condição, o contador é incrementado.

Funcionamento:

1. Executa o comando de inicialização (do contador).

2. Testa a condição de parada.

3. Se condição for verdadeira, então executa o bloco de comandos do PARA.

4. Se a condição for falsa, então sai da repetição. Executa o próximo comando após o PARA.

5. Incrementa (ou decrementa) o contador.

6. Retorna ao passo 2.

Sintaxe:

```
for (c0, condição, c1) {
      <comando>
}
```

Onde, c0 representa o comando de inicialização do contador e c1 representa o passo, isto é, o incremento ou decremento do contador. O passo pode ser maior que um, assim podemos incrementar ou decrementar o contador de dois, três, quatro etc.

Exemplo: programa que escreve os 100 primeiros pares.

A variável "i" é utilizada como contador e o seu incremento é de uma uni-

```
public static void main(String args[]) throws IOEx-
ception {
int par;
par = 0;
for (int i = 1; i<= 100; i++) {
      System.out.print(par + " - ");
      par = par + 2;
}
}
```

dade (i++). A repetição é interrompida quando o valor de "i" se torna maior que 100 (condição se torna falsa).

Resumo

Vimos nessa aula que os programas de computador são compostos por blocos de comando, sequência de instruções que tem como objetivo resolver um determinado problema. Porém, o fluxo sequencial de execução, em muitos casos, não é suficiente para termos uma solução eficiente. Nesses casos, é necessária a utilização de estruturas que permitam a quebra desse fluxo, permitindo caminhos alternativos e, quando algumas instruções devem ser executadas mais de uma vez, podemos utilizar comandos de repetição. Vimos os comandos de decisão SE (e suas variantes) e CASO, que nos permite realizar desvios no fluxo de comandos. Vimos também os comandos ENQUANTO, REPITA e PARA, que possibilitam uma repetição, controlada, de blocos de comandos, tornando nosso código mais limpo e eficiente.

Atividades de aprendizagem

1. O IBIS Futebol Clube deseja aumentar o salário de seus jogadores. O reajuste deve obedecer à seguinte tabela:

SALÁRIO ATUAL (R$)	AUMENTO
0,00 a 1.000,00	20%
1.000,01 a 5.000,00	10%
acima de 5.000,00	0%

Desenvolver um programa que leia o nome e o salário atual de um jogador, e exiba o nome, o salário atual e o salário reajustado.

2. Criar um programa que leia cinco números inteiros e imprima o maior e o menor número da lista.

3. Criar um programa que leia os limites inferior e superior de um intervalo e imprima todos os números desse intervalo. Deve-se supor que o valor do limite inferior é menor que o valor do limite superior.

Ex.: suponha que o usuário digite 4 e 9, os números do intervalo são 4, 5, 6, 7, 8 e 9.

4. Faça um programa que efetue o produto de dois números inteiros (A e B) informados pelo usuário, ou seja, A * B, através de adições (somas).

Ex.: 3 * 5 = 3 + 3 + 3 + 3 + 3 = 15

Assim terminamos a nossa terceira aula. Chegamos à metade da nossa disciplina. Ainda temos um bom caminho pela frente. Vamos seguir? Até a nossa quarta aula!

Aula 4. Programação modular

Objetivos:

- reconhecer o que são módulos; e
- saber "modularizar" um programa.

Olá! Estamos na metade da nossa disciplina. Espero que esteja gostando. E então, vamos iniciar a segunda metade dos nossos estudos?

Para iniciarmos, vamos pensar em uma situação: muitas vezes nos deparamos com problemas complexos e extremamente longos, os quais, a primeira vista, parecem não ter uma solução. Pense em como as formigas carregam seu alimento para o ninho. Algumas formigas picotam as folhas enquanto outras realizam o transporte. Pode-se perceber que algo que parece muito difícil torna-se possível com a divisão do "problema". Cada "pedaço" dessa divisão deve ser considerado como um problema menor, e o conjunto das soluções de cada um destes pequenos problemas são subsídios para a solução completa.

Figura 10: Divisão do trabalho entre as formigas.
Fonte: http://vtm.e15.cz/aktuality/prace-vseho-druhu-1-cast, acessado em 02/05/2013.

Na aula anterior, vimos que as estruturas de controle (sequência, decisão e repetição) utilizadas nos programas de computador funcionam como um bloco lógico, com início e fim. Conforme nos deparamos com problemas mais complexos, torna-se necessário quebrá-los em unidades menores que facilitam suas resoluções. Essas unidades devem ser bem estruturadas para que possamos obter resultados parciais e utilizá-los para a resolução completa do problema.

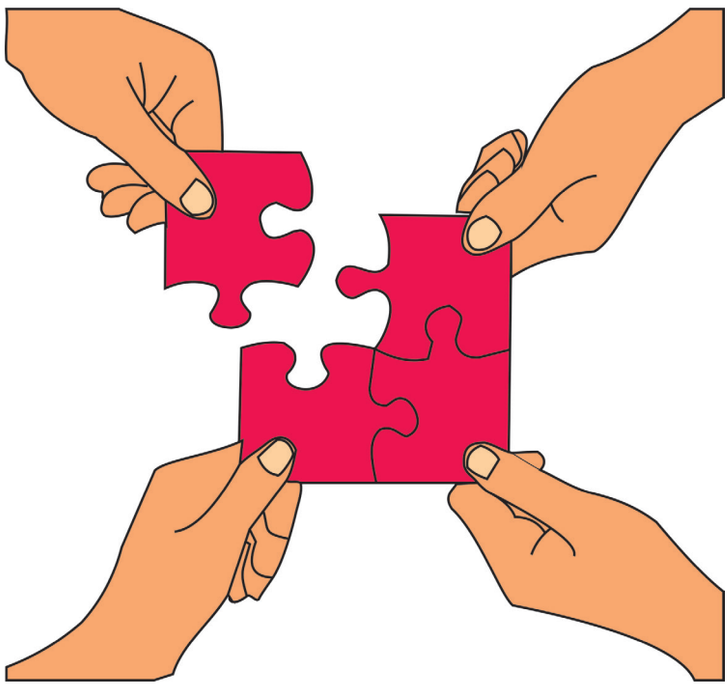


Figura 11: Resolução do problema em partes.
Fonte: http://www.sbcoaching.com.br/blog/motivacao/o-que-nao-me-destroi-fortalece-me/, acessado em 03/05/2013.

Nesta aula veremos a importância da fragmentação de ações para facilitar a resolução de problemas complexos, alcançados através da programação modular, possibilitando, ainda, a reutilização destes fragmentos (módulos) na resolução de outros problemas.

4.1 Módulo

Os módulos são unidades bem definidas, contendo blocos de instruções que visam cumprir uma tarefa. Cada módulo tem como funcionalidade retornar uma resposta a ser utilizada como parte ou subsídio da resolução do problema, portanto ele é uma parte do conjunto solução.

Através da utilização dos módulos, como ferramenta para a resolução de problemas menores, temos uma diminuição da complexidade do algoritmo, o que nos facilita a compreensão de cada componente isoladamente, além de simplificar a relação existente entre eles.

É de consenso que podemos resolver problemas complexos através da divisão de tarefas longas em tarefas menores, mais simples. Esse era o princípio da revolução industrial, em que as linhas de montagem de cada unidade eram responsáveis por uma parte do processo.

4.2 Declarando um módulo

A modularização de um programa necessita de algumas definições específicas para caracterizar esta estrutura.

Os blocos de instruções que representam os módulos devem possuir um nome específico (identificador) para podermos identificá-lo na estrutura do programa.

Além disso, devemos obedecer a algumas regras para a composição de cada módulo, pois eles devem se comportar como um pequeno programa. Essa padronização é necessária para que possamos utilizá-los nos mais variados algoritmos para resolução dos problemas. Essas regras serão vistas mais a frente, ao apresentarmos as funções e procedimentos.

A seguir, mostraremos algumas considerações sobre a construção de um módulo:

a) A construção de um módulo deve ser estruturada como a construção de um pequeno programa, o qual possui suas variáveis e produz um resultado específico;

b) Cada módulo deve possuir uma identificação e uma estrutura para poder receber informações quando acionado;

c) Os módulos poderão acionar outros módulos para a continuidade de suas execuções.

4.3 Utilizando módulos

O acionamento de um módulo acontece quando em um determinado ponto do programa o identificador do módulo é utilizado, conhecido por "chamada" ou "ativação" do módulo. Nesse ponto, a sequência de execução das instruções do programa é desviada para o módulo, em que a execução das ações deste módulo é iniciada.

Após a conclusão da execução das instruções do módulo, é retomada a execução, no programa principal, a partir do ponto de chamada.

4.4 Modularizando um programa

Vamos apresentar a resolução de um problema utilizando a modularização do algoritmo.

Considere o seguinte problema: necessitamos realizar a soma de todos os números pares entre 0 e 100. Podemos dividir esse problema em duas partes menores, uma será responsável por percorrer todos os números entre 0 e 100, chamada "Percorrer()", e a outra verificará se o número é par, chamada "Par()". Assim, temos dois pequenos problemas: "Percorrer()", que deverá possuir instruções para identificar cada um dos números no intervalo, e "Par(), que fará a verificação se o número é par. No módulo "Percorrer()" iremos chamar o módulo "Par()" para identificar se o número atual é par, e sendo verdade, realizamos a soma. Dessa forma, teríamos os seguintes módulos:

```
Par(){
      ações para verificar se o valor é par;
}
Percorrer(){
      ações para percorrer cada elemento;
      para cada elemento chamar Par();
      somar o elemento se for par;
}
```

Vamos ver abaixo as vantagens da "modularização":

Vantagens

- Programas mais simples de escrever: a divisão em pequenas partes torna

a criação do código mais simples.

- Código mais claro: os programas escritos em módulos tornam-se mais fáceis de serem lidos.

- Correções mais fáceis: os programas modulares são mais fáceis de modificar.

- Reutilização: os módulos podem ser reaproveitados para a resolução de outros problemas.

Resumo

À medida que avançamos no curso, os problemas tornam-se cada vez mais complexos, tornando suas resoluções cada vez mais complicadas. Tratar esses problemas sem uma abordagem eficiente poderá se transformar numa tarefa impossível de se realizar. Vimos que dividir um problema longo e complexo em tarefas menores poderá facilitar a resolução do todo. Problemas menores tendem a ser mais simples e também mais fáceis de ser compreendidos.

A modularização dos programas é uma das técnicas de programação que dão maior clareza ao código, possibilitam que blocos modularizados possam ser utilizados na resolução de outros problemas e possibilitam, ainda, uma facilidade maior na manutenção do código.

Atividades de aprendizagem

1. Utilizando o exemplo de modularização apresentado anteriormente, descreva os módulos necessários para resolução do seguinte problema: dados dois números, identificar o maior e verificar se o mesmo é par ou ímpar.

2. Ainda utilizando as definições acima, resolva o seguinte problema: dada uma sequência de números, arranjá-los em ordem crescente e decrescente.

3. Dado um número, identificar se ele é primo, se é par ou ímpar e se é um ano bissexto.

Estamos entrando na reta final da nossa disciplina. Faltam apenas duas aulas para chegarmos ao final. Vamos para a nossa quinta aula? Até lá!

Aula 5. Funções e procedimentos

Objetivos:

- conceituar e utilizar funções e procedimentos;
- utilizar variáveis locais e globais;
- utilizar parâmetros em funções e procedimentos; e
- diferenciar funções e procedimentos.

Vamos iniciar a nossa penúltima aula. Veremos, aqui, como modularizar um programa através da utilização dos conceitos de funções e procedimentos. Verificaremos como construir essas estruturas de modo a tornar nosso programa mais eficiente e limpo. Além disso, como aplicar os conceitos envolvidos para passar informações a esses módulos e como obter seus resultados.

Em muitas situações do nosso dia a dia nos deparamos com os conceitos de funções e procedimentos: o funcionário da empresa tem determinada função; para trocar o pneu do carro realizamos certos procedimentos. Podemos generalizar os conceitos de função e procedimento como sendo uma atividade (ações) específica realizada por determinado elemento (pessoa, produto, etc.) para a conclusão de um objetivo em um sistema.

Como vimos anteriormente, a divisão de tarefas é uma técnica da modularização. Essa divisão facilita a resolução de um problema e permite que esses módulos possam ser utilizados em resoluções de outros problemas.

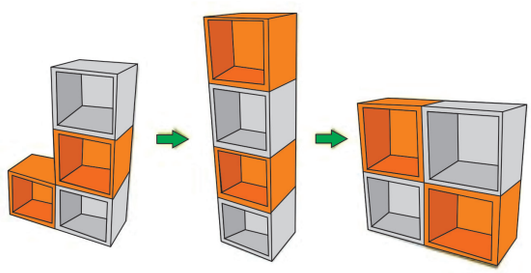


Figura 12: Utilização de módulos em soluções diversas.
Fonte: www.redninjapress.com/2009/07/o-rpg-de-amanha/, acessado em: 07/05/2013.

Vamos conhecer como as variáveis são tratadas em um programa JAVA modularizado.

5.1 Variável local e global

As variáveis são especificadas de acordo com o escopo (local onde são criadas) de sua declaração. Variáveis declaradas dentro de um método (função ou procedimento) são nomeadas variáveis locais e são conhecidas (podem ser acessadas) apenas dentro do método. As variáveis declaradas na classe, que constitui o programa, são nomeadas globais e podem ser acessadas dentro de qualquer método da classe.

```
public class Variaveis {
      //Declaração de uma variável Global
      String var_global = "Olá";
public static void main(String args[]) throws IOEx-
ception {
            Scanner entrada = new Scanner(System.
in);
            //Declaração de uma variável Local
            String var_local;
            System.out.print("Digite algo:");
            var_local = entrada.nextLine();
            System.out.print(var_global + var_lo-
cal);
}
}
```

5.2 Procedimentos

Os procedimentos são partes de um programa (módulo), compostos por um conjunto de comandos, utilizados para realizar a resolução de uma determinada tarefa.

Os procedimentos são independentes e normalmente são utilizados para criar um produto.

Procedimentos em JAVA são declarados pelo comando "void" seguido do nome do procedimento.

Sintaxe:

```
visibilidade void NomeDoProcedimento(<parâmetros>)
{
<comandos>
}
```

Onde:

- **visibilidade:** será melhor explicada na disciplina Orientação a Objeto. Por enquanto utilizaremos a palavra reservada "public";
- **void:** especifica um procedimento;
- **Nome do procedimento:** conjunto de caracteres que identifica o procedimento;
- **<parâmetros>:** conjunto de variáveis que receberão as informações passadas ao procedimento. Veremos de forma mais esclarecida a utilização de parâmetros no decorrer deste capítulo;
- **<comandos>:** bloco de comandos executados pelo procedimento.

Acesse o *site* para maiores informações sobre métodos em JAVA: **http://www.devmedia.com.br/trabalhando-com-metodos-em-java/25917**

A chamada de um procedimento é feita através da citação do seu nome seguido opcionalmente de seu argumento inicial entre parênteses.

Em Java, funções e procedimentos são casos especiais de métodos.

5.2.1 Parâmetros

Os parâmetros (argumentos) são a porta de entrada dos procedimentos e funções. Através deles podemos passar uma informação a qual será utilizada no processo (bloco de comandos). Os parâmetros se comportam como variáveis no procedimento.

Sintaxe:

```
(TipoDeDado nomeDoParâmetro1, TipoDeDado nomeDoParâmetro2, ...)
```

Onde:

- TipoDeDado: tipo de dado referente ao valor atribuído ao parâmetro.
- Os parâmetros são opcionais e quando há a necessidade de utilizar mais de um, cria-se uma lista separada por vírgula.

Exemplo: procedimento para escrever uma mensagem na console.

```
import java.util.Scanner;

public class Procedimento {
public static void main(String args[]) throws IOEx-
ception {
      Scanner entrada = new Scanner(System.in);
      String texto;
      System.out.print("Digite algo:");
      texto = entrada.nextLine();
      Imprime(entrada);
}

public static void Imprime (String texto) {
      System.out.print(texto);
}
}
```

Nesse exemplo, o procedimento (método) "Imprime()' recebe a informação do texto a ser escrito na console através do argumento "String texto".

5.3 Funções

O que são funções?

Funções são instrumentos que têm como objetivo retornar um valor ou uma informação.

Uma função é definida na linguagem de programação JAVA de maneira semelhante à conhecida na Matemática. As funções recebem valores que são passados como argumentos (parâmetros) e, após um processamento, retorna um valor como resultado.

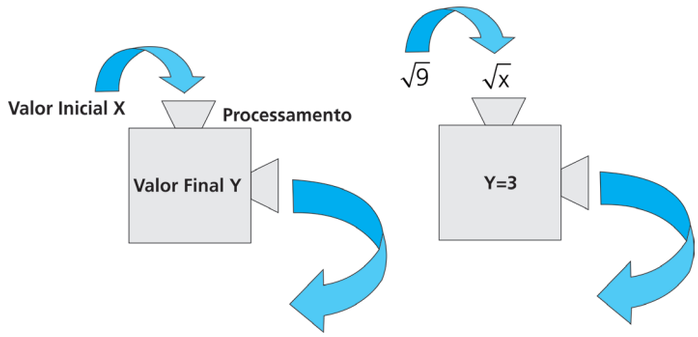


Figura 13: Funções
Fonte: autor.

A chamada de uma função é feita através da citação do seu nome seguido opcionalmente de seu argumento inicial entre parênteses.

Funções em JAVA são declaradas por um tipo de dado para retorno do valor seguido do nome da função.

Sintaxe:

Onde:

```
visibilidade TipoDeRetorno
NomeDaFuncao(<parâmetros>)
{
<comandos>
return valor;
}
```

- visibilidade: será melhor explicada na disciplina Orientação a Objeto. Por enquanto utilizaremos a palavra reservada "public";
- TipodeRetorno: especifica o tipo de dado da informação retornada pela função;
- NomeDaFuncao: conjunto de caracteres que identifica a função;
- <parâmetros>: conjunto de variáveis que receberão as informações passadas à função;
- <comandos>: bloco de comandos executados pelo procedimento;
- return: conclui a função e retorna a informação obtida.

Exemplo: calcular o fatorial de um número

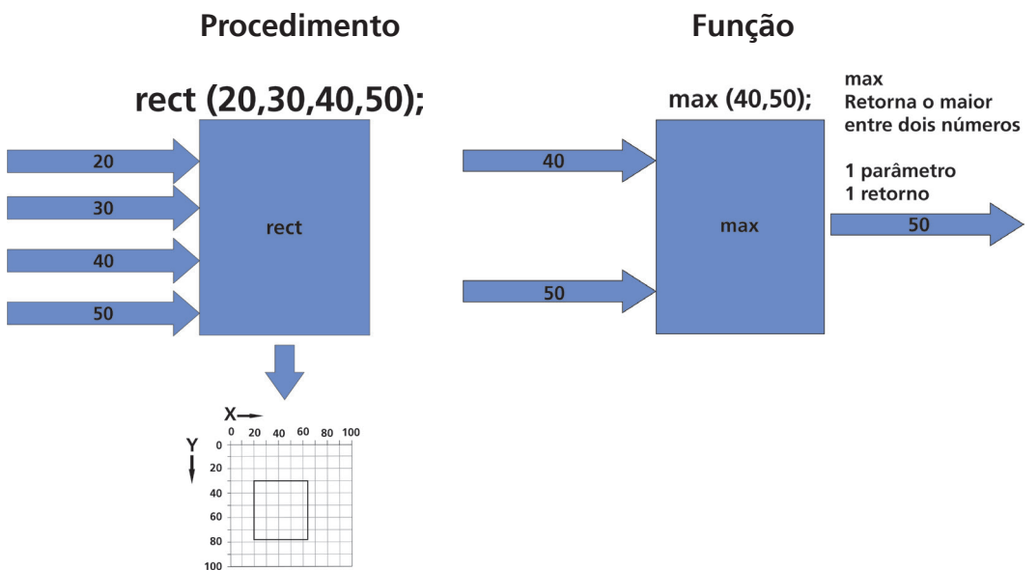
```
import java.util.Scanner;

public class Funcao {
public static void main(String args[]) throws IOEx-
ception {
      Scanner entrada = new Scanner(System.in);
      int numero;
      System.out.print("Digite um número positi-
vo:");
      numero = entrada.nextLine();
       System.out.println("O factorial é: " +
Fatorial(numero));
}

public static int Fatorial(int num) {
      int f = 1;
      for (int I = 1; I <= num; i++)
            f = f * i;
      return f;
}
}
```

5.3.1 Funções X Procedimentos

A diferença básica entre função e procedimento é que a função retorna um ou mais valores e o procedimento simplesmente executa uma série de comandos, sem retornar valores.



Executa um conjunto de comandos para resolução de um problema.

Executa um conjunto de comandos e retorna um valor solicitado.

Resumo

A divisão de uma tarefa em tarefas menores (módulos) facilita a resolução de um problema complexo e extenso. Essa divisão, em programação, é conhecida como modularização. Vimos nessa aula como realizar a modularização de um programa através da utilização de funções e procedimentos. Identificamos as diferenças entre as funções e procedimentos. Vimos também como passar informações para serem utilizadas internamente nessas estruturas através dos conceitos de parâmetros.

Atividades de aprendizagem

1. Crie um procedimento que receba um valor inteiro e positivo entre 0 e 10 e escreva na console seu valor por extenso, isto é, para 0 escreva "zero", para 1 escreva "um" e assim respectivamente para os demais.

2. Implemente uma função que verifique se um valor é perfeito. Um valor é considerado perfeito quando ele é igual à soma dos seus divisores excetuando ele próprio. (Ex.: 6 é perfeito, 6 = 1 + 2 + 3, que são seus divisores). A função deve retornar um valor booleano.

3. Faça uma função que receba a idade de um atleta e retorne sua categoria, conforme a tabela abaixo:

Idade	Categoria
5 a 7 anos	Infantil A
8 a 10 anos	Infantil B
11-13 anos	Juvenil A
14-17 anos	Juvenil B
Maiores de 18 anos (inclusive)	Adulto

4. Faça uma função que receba um valor inteiro e verifique se o mesmo é par ou ímpar. A função deve retornar "par" quando o valor for par e "ímpar" quando for ímpar.

Assim terminamos o nosso quinto e penúltimo encontro. Espero que esteja aproveitando a nossa disciplina. Até a nossa sexta e última aula!

Aula 6. Recursividade

Objetivos:

- utilizar a técnica da recursividade; e
- identificar as vantagens e desvantagens da técnica da recursividade.

Estamos chegando ao final da nossa disciplina. Com ela, você já teve contato com uma série de conceitos sobre técnicas de programação. Agora falta pouco. Vamos em frente?

Pois bem, você já teve a curiosidade de colocar um espelho na frente de outro? Vemos uma infinidade de imagens refletidas. As imagens repetidas que aparecem quando os dois espelhos são dispostos dessa maneira são denominadas recursivas.

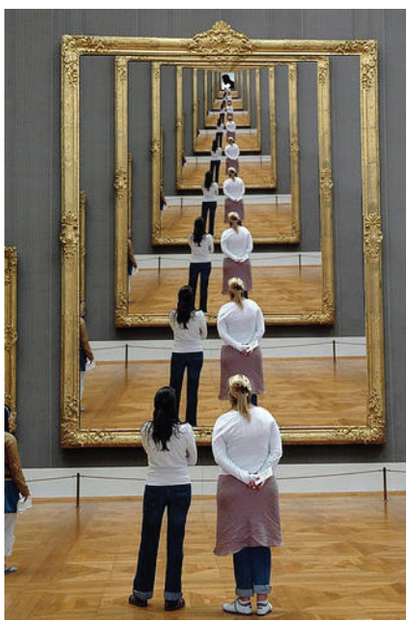


Figura 14: Recursão em espelho
Fonte: Internet - http://bluehawk.monmouth.edu/rclayton/web-pages/s11-503/recursion.jpg, acessado em 20/05/2013.

A recursividade pode ser encontrada em muitos casos matemáticos, em que uma fórmula é aplicada repetidas vezes para resolver um problema.

Veja na imagem abaixo, o fatorial de 5 (5!) é calculado aplicando-se a fórmula fatorial(n) = n * fatorial(n-1), no qual fatorial(0) = 1.

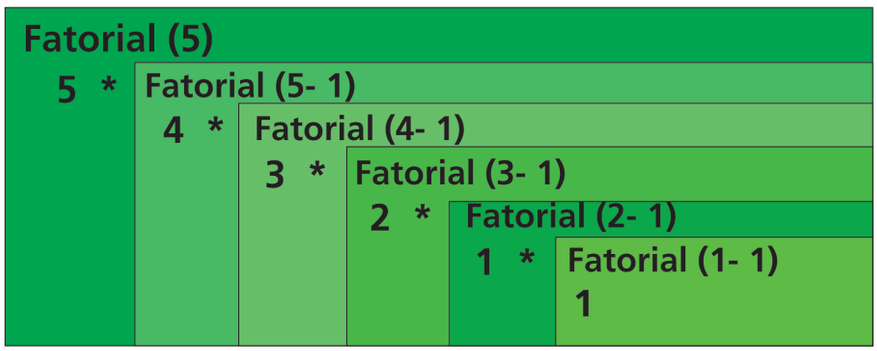


Figura 15: Recursividade em função fatorial
Fonte: Internet - http://olamundo0.files.wordpress.com/2010/04/fatorial.jpg?w=595, acessado em 20/05/2013.

Assim temos:

```
fatorial(5) = 5 * fatorial(4)
fatorial(4) = 4 * fatorial(3)
fatorial(3) = 3 * fatorial(2)
fatorial(2) = 2 * fatorial(1)
fatorial(1) = 1 * fatorial(0)
fatorial(0) = 1
fatorial(1) = 1 * 1 = 1
fatorial(2) = 2 * 1 = 2
fatorial(3) = 3 * 2 = 6
fatorial(4) = 4 * 6 = 24
fatorial(5) = 5 * 24 = 120
```

Como foi apresentado na aula de modularização, um módulo (função ou procedimento) pode ser ativado (chamado) por outro módulo. Veremos nesta aula, um caso particular de módulos, a recursividade, isto é, quando um módulo ativa ou faz uma chamada a si mesmo.

Em programação, a recursividade ocorre nas situações em que se tem uma função que chama (aciona) ela mesma.

Os algoritmos recursivos são funções que consistem em dividir um problema em problemas menores, subproblemas, semelhantes, "dividir para conquistar".

6.1 Recursão

Então, o que é recursão?

A recursão é uma técnica que consiste em dividir problemas maiores em vários problemas menores, mais fáceis de serem resolvidos, os quais fornecem a resolução para o problema original.

Abordagem geral:

1. Dividir um problema em pequenos problemas semelhantes, subproblemas;

2. Repetir até que os subproblemas se tornem triviais;

a) Problema "trivial" pode ser resolvido sem cálculo, um cálculo simples ou por "força bruta";

3. Recombinar as soluções dos subproblemas em uma solução para o problema original.

Onde:

- Encontrada a solução do subproblema "trivial" no passo 2, vamos chamá-lo de "função base", o processo recursivo termina.

- Recursividade dos passos 1 e 3:

  - Simplifica um problema em pequenos subproblemas similares, "função recursiva". Garantir que a função decresça a cada passo de repetição, permitindo que a solução trivial seja atingida.
  - Recursivamente aplica-se o algoritmo para os subproblemas.
  - Combina as soluções dos subproblemas numa solução completa.

Exemplo: somar os números inteiros menores ou igual a 5.

1. Encontrar a função recursiva: f(n) = n + f(n-1)

2. Encontrar a função base: f(0) = 0

Assim temos:

```
f(5) = 5 + f(4)
f(4) = 4 + f(3)
f(3) = 3 + f(2)
f(2) = 2 + f(1)
f(1) = 1 + f(0)
f(0) = 0
```

Recombinar as soluções dos subproblemas:

```
f(1) = 1 + 0 = 1
f(2) = 2 + 1 = 3
f(3) = 3 + 3 = 6
f(4) = 4 + 6 = 10
f(5) = 5 + 10 = 15
```

6.2 Algoritmo recursivo

A seguir apresentaremos um algoritmo genérico para aplicação de recursividade em um programa:

- se o problema em questão é pequeno (função base),

  então, resolva-o;

- senão,

  reduza-o a um problema menor (subproblema), semelhante (função recursiva),

  aplique o algoritmo ao subproblema e

  volte ao problema original.

Exemplo: considerando o problema do exemplo anterior, temos:

```
função Soma(n) {
Se (n <= 0) //função base
      então retorna 0;
Senão
      retorna n + Soma(n-1); //recursividade
}
```

A solução do problema acima depende de se encontrar:

- função base: f(0) = 0, então, quando temos n igual a 0, o resultado é 0. Nesse momento devemos interromper a recursão "Se (n<=0)";
- função recursiva: f(n) = n + f(n-1).

Cada vez que a função Soma() é invocada, o problema torna-se menor até atingir a solução trivial.

Vamos ver abaixo as vantagens e desvantagens da utilização da recursividade.

Vantagens

- Um programa recursivo é mais simples (menor que a versão iterativa) e elegante.
- Pode ser aplicada nas mais variadas situações.
- Exibe com maior clareza o processo empregado.

Desvantagens

- Custo de cálculo, na maioria dos casos é mais lento que a versão iterativa.
- Erros de design sutis e difíceis de serem encontrados.
- Dificuldade em encontrar a recursão.
- Exige mais espaço de memória.

Resumo

Vimos nessa aula que problemas complexos, extensos, podem ser resolvidos através da divisão em problemas menores, subproblemas, com as mesmas características. Essa técnica é denominada recursividade e é aplicada através da utilização de funções (função recursiva) que são chamadas por elas mesmas (recursão). Cada vez que a função é invocada, o problema torna-se menor até atingir uma solução trivial, que possui um valor conhecido ou de fácil cálculo, chamada de função base.

Atividades de aprendizagem

1. Considere a seguinte função recursiva e o valor de x sempre menor que 100:

Descreva o que essa função calcula.

2. Desenvolva um programa, através de recursividade, para resolver o problema proposto no exemplo dessa aula. O usuário deverá entrar com um

```
int Funcao(int x) {
if ( x > 100 )
      return 0;
else
      return x + Funcao(x + 1);
}
```

valor inteiro e o programa deverá calcular a soma de todos os inteiros entre 0 e o valor informado (inclusive).

3. Implemente um algoritmo, utilizando recursividade, para calcular o fatorial de um número informado pelo usuário. Lembre-se de que o fatorial é calculado através do produto de todos os inteiros positivos menores ou iguais a n, fatorial de 0 é 1.

Assim terminamos a nossa sexta e última aula. Espero que tenha tirado bom proveito dela. Siga a leitura do caderno, pois tenho mais algumas palavras para dizer a você. Um grande abraço!

Palavras Finais

Caro(a) estudante,

Meus parabéns por completar mais esta etapa. Chegamos ao final da disciplina. Durante nossa caminhada adquirimos conhecimento, pudemos realizar trocas de experiências e de ideias. Estamos mais preparados para seguir, para enfrentar novos desafios. Este é um ponto de partida para um aprendizado maior que virá com o tempo, com a experiência.

Sabemos que não é fácil chegar até aqui, acredito que em muitas vezes você se desanimou, as dificuldades e a fadiga foram obstáculos cruéis, mas sua determinação e persistência fizeram com que você vencesse.

Não podemos esquecer que a busca pelo conhecimento não termina aqui, ela é cíclica, portanto, devemos continuar procurando novos desafios. Como criar um site de relacionamento? Qual estrutura é mais adequada? Como manipular tantas informações? Esses são apenas alguns dos problemas enfrentados pelos programadores, e podemos sim encará-los e resolvê-los. Para isso, temos que manter nossa disposição em continuar.

Guia de Soluções

Aula 1

1. Dado o algoritmo abaixo, "Calcular Valor Pedido", identifique os dados de entrada, qual o processamento e os dados de saída:

- Leia o código do item.
- Leia o valor do item.
- Leia a quantidade do item.
- Calcule o valor total do pedido (quantidade * valor do item).
- Escreva o valor total do pedido.

a) Dados de entrada: código do item, valor do item e quantidade do item

b) Processamento: quantidade * valor do item

c) Dados de saída: valor total do pedido

2. Crie um programa que leia o nome do usuário e sua idade, depois escreva os valores obtidos.

```
import java.util.Scanner;

public class Ex02 {
	public static void main(String[] args) {
		String nome;
		int idade = 0;
		Scanner s = new Scanner(System.in);
		System.out.println("Digite o seu nome:
");
		nome = s.next();
		System.out.print("Digite o sua idade:
");
		idade = s.nextInt();
		System.out.println("Seu nome é: " +
nome);
		System.out.print("Sua idade é: " +
idade);
	}
}
```

3. Faça um programa que leia dois valores inteiros e armazene cada um em uma variável, depois troque os valores das duas variáveis entre si. Como saída, escreva os valores lidos.

```
import java.util.Scanner;

public class Ex03 {
      public static void main(String[] args) {
            int x, y, aux;
            Scanner s = new Scanner(System.in);

            System.out.println("Digite o primeiro
valor: ");
            x = s.nextInt();
            System.out.print("Digite o segundo val-
or: ");
            y = s.nextInt();
            aux = x;
            x = y;
            y = aux;
            System.out.println("Primeiro valor: " +
y);
            System.out.print("Segundo valor: " + x);
      }
}
```

Aula 2

1. Informe qual o valor da variável resultado, em cada linha, após a execução das seguintes operações:

a) resultado = 3 + 6; x = 2; y = 3;

resultado = 9

b) resultado = resultado / y – x;

resultado = 1

c) resultado = 4; x = 2;

resultado = 4

d) resultado = resultado * x;

resultado = 8

2. Faça um programa que calcule a média de três valores informados pelo usuário.

```
import java.util.Scanner;

public class Ex0202 {
	public static void main(String[] args) {
		int x, y, z;
		double media;
		Scanner s = new Scanner(System.in);

		System.out.println("Digite o primeiro
valor: ");
		x = s.nextInt();
		System.out.print("Digite o segundo val-
or: ");
		y = s.nextInt();
		System.out.print("Digite o terceiro val-
or: ");
		z = s.nextInt();
		media = (x + y + z) / 3;
		System.out.println("A média é: " + me-
dia);
	}
}
```

3. Faça um programa que, dada uma temperatura em graus Celsius, converta-a para Fahrenheit, conforme a fórmula:

°F = (1.8 * °C) + 32

```
import java.util.Scanner;

public class Ex0203 {
	public static void main(String[] args) {
		double c, f;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe a tempera-
tura em Celcius: ");
		c = s.nextDouble();
		f = (1.8 * c) + 32;
		System.out.println("A temperatura em
Fahrenheit é: " + f);
	}
}
```

Aula 3

1. O IBIS Futebol Clube deseja aumentar o salário de seus jogadores. O reajuste deve obedecer a seguinte tabela:

SALÁRIO ATUAL (R$)	AUMENTO
0,00 a 1.000,00	20%
1.000,01 a 5.000,00	10%
acima de 5.000,00	0%

Desenvolver um programa que leia o nome e o salário atual de um jogador, e exiba o nome, o salário atual e o salário reajustado.

```
import java.util.Scanner;

public class Ex0301 {
      public static void main(String[] args) {
            double salario, sal_ajustado;
            String nome;
            Scanner s = new Scanner(System.in);

            System.out.println("Informe o nome do
funcionário: ");
            nome = s.next();
            System.out.println("Informe o salário do
funcionário: ");
            salario = s.nextDouble();
            if (salario <= 1000)
                  sal_ajustado = salario * 1.2;
            else if ( salario > 1000 && salario <
5000)
                  sal_ajustado = salario * 1.1;
            else
                  sal_ajustado = salario;
            System.out.println("O nome do fun-
cionário é: " + nome);
            System.out.println("O salário é: " +
salario);
            System.out.println("O salário ajustado
é: " + sal_ajustado);
       }
}
```

2. Criar um programa que leia cinco números inteiros e imprima o maior e o menor número da lista.

Para a solução do problema, não utilizaremos vetores, os quais serão apresentados na disciplina Estrutura de Dados.

```
import java.util.Scanner;

public class Ex0302 {
      public static void main(String[] args) {
            int x1, x2, x3, x4, x5;
            int maior, menor;
            Scanner s = new Scanner(System.in);

            System.out.println("Informe o primeiro
número: ");
            x1 = s.nextInt();
            System.out.println("Informe o segundo
número: ");
            x2 = s.nextInt();
            System.out.println("Informe o terceiro
número: ");
            x3 = s.nextInt();
            System.out.println("Informe o quarto
número: ");
            x4 = s.nextInt();
            System.out.println("Informe o quinto
número: ");
            x5 = s.nextInt();
            //Verificar o maior
            maior = x1;
            if (x2 > maior)
                  maior = x2;
            if (x3 > maior)
                  maior = x3;
            if (x4 > maior)
                  maior = x4;
            if (x5 > maior)
                  maior = x5;
            //Verificar o menor
            menor = x1;
            if (x2 < menor)
                  menor = x2;
            if (x3 < menor)
                  menor = x3;
            if (x4 < menor)
                  menor = x4;
            if (x5 < menor)
                  menor = x5;
            System.out.println("O maior valor é: " +
maior);
            System.out.println("O menor valor é: " +
menor);
      }
}
```

3. Criar um programa que leia os limites inferior e superior de um intervalo e imprima todos os números desse intervalo. Deve-se supor que o valor do limite inferior é menor que o valor do limite superior.

Ex.: suponha que o usuário digite 4 e 9, os números do intervalo são 4, 5, 6, 7, 8 e 9.

```
import java.util.Scanner;

public class Ex0303 {
      public static void main(String[] args) {
            int inferior, superior;
            Scanner s = new Scanner(System.in);

            System.out.println("Informe o limite in-
ferior do intervalo: ");
            inferior = s.nextInt();
            System.out.println("Informe o limite su-
perior do intervalo: ");
            superior = s.nextInt();
            while(inferior <= superior){
                  System.out.println(inferior);
                  inferior = inferior + 1;
            }
      }
}
```

4. Faça um programa que efetue o produto de dois números inteiros (A e B) informados pelo usuário, ou seja, A * B, através de adições (somas).

Ex.: 3 * 5 = 3 + 3 + 3 + 3 + 3 = 15

```
import java.util.Scanner;

public class Ex0304 {
	public static void main(String[] args) {
		int val1, val2;
		int result, cont;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe o primeiro
valor: ");
		val1 = s.nextInt();
		System.out.println("Informe o segundo
valor: ");
		val2 = s.nextInt();
		result = 0;
		cont = 0;
		while(cont < val2){
			result = result + val1;
			cont = cont + 1;
		}
		if (val2 == 0)
			result = 1;
		System.out.println("O resultado da mul-
tiplicação é: " + result);
	}
}
```

Aula 4

1. Utilizando o exemplo de modularização apresentado anteriormente, descreva os módulos necessários para resolução do seguinte problema: dados dois números, identificar o maior e verificar se o mesmo é par ou ímpar.

Aqui temos duas situações: identificar o maior e verificar se o mesmo é par ou ímpar

```
Principal(){
	chama Maior();
	chama Par() informando o valor de maior;
	se maior for par escreve “par”;
	caso contrário escreve “impar”;
}
Maior(){
	ações para verificar o maior valor;
}
Par(){
	ações para verificar se é par;
}
```

Podemos ter outras soluções semelhantes, também válidas, já que não temos uma regra específica para escrevermos esses comandos.

2. Ainda utilizando as definições acima, resolva o seguinte problema: dada uma sequência de números, arranjá-los em ordem crescente e decrescente.

```
Principal(){
      chama Crescente();
      escreve os números;
      chama Decrescente();
      escreve os números;
}
Crescente(){
      ações para arranjar os números em ordem cres-
cente;
}
Crescente(){
      ações para arranjar os números em ordem de-
crescente;
}
```

3. Dado um número, identificar se ele é primo, se é par ou ímpar e se é um ano bissexto.

```
Principal(){
      chama Primo();
      escreve se é primo;
      chama Par();
      escreve "par" se é par;
      caso contrário escreve "ímpar";
      chama Bissexto();
      escreve se é bissexto;
}
Primo(){
      ações para verificar se o número é primo;
}
Par(){
      ações para verificar se o número é par;
}
Bissexto(){
      ações para verificar se o número é bissexto;
}
```

Aula 5

1. Crie um procedimento que receba um valor inteiro e positivo entre 0 e 10 e escreva na console seu valor por extenso, isto é, para 0 escreva "zero", para 1 escreva "um" e assim respectivamente para os demais.

```
import java.util.Scanner;

public class Ex0501 {
	public static String Extenso(int num){
		String extenso;
		switch(num){
		case 0: extenso = "Zero"; break;
		case 1: extenso = "Um"; break;
		case 2: extenso = "Dois"; break;
		case 3: extenso = "Três"; break;
		case 4: extenso = "Quatro"; break;
		case 5: extenso = "Cinco"; break;
		case 6: extenso = "Seis"; break;
		case 7: extenso = "Sete"; break;
		case 8: extenso = "Oito"; break;
		case 9: extenso = "Nove"; break;
		case 10: extenso = "Dez"; break;
		default: extenso = "número
inválido";break;
		}
		return extenso;
	}

	public static void main(String[] args) {
		int num;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe um número de
0 a 10: ");
		num = s.nextInt();
		System.out.println("O extenso é: " +
Extenso(num));
	}
}
```

2. Implemente uma função que verifique se um valor é perfeito. Um valor é considerado perfeito quando ele é igual a soma dos seus divisores excetuando ele próprio. (Ex.: 6 é perfeito, 6 = 1 + 2 + 3, que são seus divisores). A função deve retornar um valor booleano.

```
import java.util.Scanner;

public class Ex0502 {
	public static Boolean Perfeito(int num){
		Boolean perfeito = false;
		int soma = 0;
		for(int i=1; i<num; i++){
			if(num%i==0)
				soma = soma + i;
		}
		if (soma == num)
			perfeito = true;
		return perfeito;
	}

	public static void main(String[] args) {
		int num;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe um número:
");
		num = s.nextInt();
		if(Perfeito(num))
				System.out.println("O número
é perfeito.");
		else
			System.out.println("O número não é
perfeito.");
	}
}
```

3. Faça uma função que receba a idade de um atleta e retorne sua categoria, conforme a tabela abaixo:

Idade	Categoria
5 a 7 anos	Infantil A
8 a 10 anos	Infantil B
11-13 anos	Juvenil A
14-17 anos	Juvenil B
Maiores de 18 anos (inclusive)	Adulto

```java
import java.util.Scanner;

public class Ex0503 {
	public static String Categoria(int idade){
		String categoria = "inválida";
		if(idade >= 5 && idade <= 7)
			categoria = "infantil A";
		else if(idade >= 8 && idade <= 10)
			categoria = "infantil B";
		else if(idade >= 11 && idade <= 13)
			categoria = "juvenil A";
		else if(idade >= 14 && idade <= 17)
			categoria = "juvenil B";
		else if(idade >= 18)
			categoria = "adulto";
		return categoria;
	}

	public static void main(String[] args) {
		int idade;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe a idade do
atleta: ");
		idade = s.nextInt();
		System.out.println("Sua categoria é: " +
Categoria(idade));
	}
}
```

4. Faça uma função que receba um valor inteiro e verifique se o mesmo é par ou ímpar. A função deve retornar "par" quando o valor for par e "ímpar" quando for ímpar.

```
import java.util.Scanner;

public class Ex0504 {
	public static String ParImpar(int num){
		String result;
		if(num%2==0)
			result = "par";
		else
			result= "ímpar";
		return result;
	}

	public static void main(String[] args) {
		int num;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe um número:
");
		num = s.nextInt();
		System.out.println("O número é: " +
ParImpar(num));
	}
}
```

Aula 6

1. Considere a seguinte função recursiva e o valor de x sempre menor que 100:

```
int Funcao(int x) {
if ( x > 100 )
	return 0;
else
	return x + Funcao(x + 1);
}
```

Descreva o que essa função calcula.

Calcula o somatório de todos os números inteiros no intervalo entre x (valor informado) e 100.

2. Desenvolva um programa, através de recursividade, para resolver o problema proposto no exemplo dessa aula. O usuário deverá entrar com um valor inteiro e o programa deverá calcular a soma de todos os inteiros entre 0 e o valor informado (inclusive).

```
import java.util.Scanner;

public class Ex0602 {
	public static int Somatorio(int x){
		if ( x <= 0 )
			return 0;
		else
			return x + Somatorio(x - 1);
	}

	public static void main(String[] args) {
		int num;
		Scanner s = new Scanner(System.in);

		System.out.println("Informe um número:
");
		num = s.nextInt();
		System.out.println("O resultado é: " +
Somatorio(num));
	}
}
```

Referências

ARAÚJO, Everton Coimbra. **Algoritmos**: Fundamento e Prática. 3 ed. São Paulo: Visual Books, 2007.

CARBONI, Irenice de Fátima. **Lógica de Programação**. São Paulo: Thomson, 2003.

FORBELLONE, André Luiz Villar. **Lógica de programação:** a construção de algoritmos e estrutura de dados. 3 ed. São Paulo: Brochura, 2005.

LOPES, Anita; GARCIA, Guto. **Introdução à Programação:** 500 Algoritmos Resolvidos. Rio de Janeiro: Campus, 2002.

SOARES, Márcio Vieira; GOMES, Marcelo Marques; Souza, Marco Antônio. **Algoritmos e Lógica de Programação**. 2. ed. São Paulo: Cengage Learning, 2011.

Bibliografia Básica

CORMEN, Thomas H.; RIVEST, Ronald L.; STEIN, Clifford; LEISERSON, Charles E. **Algoritmos**: teoria e prática. 3.ed. São Paulo: Érica, 2012.

FARRELL, Joyce. **Lógica e Design de Programação**. São Paulo: Cengage Learning, 2009.

MANZANO, José Augusto N G; OLIVEIRA, Jayr Figueiredo de. **Algoritmos**: Lógica para Desenvolvimento de Programação de Computadores. 22 ed. São Paulo. Editora Érica, 2009.

Currículo dos Professores-autores

José Marcio Benite Ramos

Bacharel em Ciências da Computação pela Universidade Federal de São Carlos - UFSCar.

Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina - UFSC.

Professor Titular responsável pelas disciplinas de Estrutura de Dados, Programação Web, Sistemas Comerciais da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor Titular responsável pelas disciplinas de Programação Web da Faculdade de São Mateus – FSM-RO.

Analista de Sistemas e diretor da divisão Esc. de Porto Velho do Serviço Federal de Processamento de Dados SERPRO.

Liluyoud Cury de Lacerda

Bacharel em Ciências da Computação pela Universidade Federal de São Carlos - UFSCar.

Mestre em Ciências da Computação pela Universidade Federal de Santa Catarina - UFSC.

Coordenador do Curso de Sistemas de Informação da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Coordenador do Curso de Sistemas para Internet da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Coordenador do Curso de Pós-Graduação em Desenvolvimento Web da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor Titular responsável pelas disciplinas de Algoritmos, Padrões de Pro-

jeto e Inteligência Artificial da Faculdade de Ciências Administrativas e Tecnologia – FATEC-RO.

Professor Titular responsável pelas disciplinas de Programação Orientada a Objetos da Faculdade de São Mateus – FATESM-RO.

Analista Programador do Ministério Público do Estado de Rondônia - MP/RO.

Sara Luize Oliveira Duarte

Mestre em Gestão e Desenvolvimento Regional pela Universidade de Taubaté (UNITAU).

Pós-Graduação em Desenvolvimento Web e Metodologia do Ensino Superior. Graduada em Processamento de Dados, ambos pela FATEC-RO.

Professora das Faculdades FATESM e FATEC. Professora Titular responsável pela disciplina Informática Básica para Trabalhos Acadêmicos, ofertada na modalidade semipresencial na Faculdade de Tecnologia São Mateus – FATESM.

Supervisora de Tecnologia do Laboratório de Educação a Distância - LED e Supervisora das Salas Virtuais da modalidade Semipresencial da FATESM.

Professora Conteudista de alguns Guias de Estudos, como Informática Básica para Trabalhos Acadêmicos, Metodologia da Pesquisa Científica, Como estudar na EAD, Informática Básica e outros.