EAD

EDUCAÇÃO A DISTÂNCIA

UNIDADE I

Paradigmas de Linguagens

Prof. Me. Luiz Forçan

Por que os Paradigmas de Linguagens?

- Paradigma é um exemplo ou um padrão a ser seguido, que tem o significado de modelo ou exemplo.
- É uma palavra utilizada para representar uma maneira de pensar ou agir, em determinada época ou em um contexto específico.
- Ou seja, é um modelo ou padrão a ser seguido pela maioria das pessoas.
- Para um desenvolvedor de *software* é o modo como ele, de maneira organizada, estrutura os comandos de determinada linguagem de computador, com o objetivo de gerar um *software* capaz de executar uma tarefa ou uma atividade para resolver um problema para alguém.
    - O desenvolvedor ou programador pode adotar um paradigma de linguagem, como uma metodologia, para desenvolver um programa de computador.
    - Então, a decisão de qual paradigma deve ser utilizado é feita pelo ponto de vista da realidade ou da maneira como o desenvolvedor atua sobre ela, estabelecendo uma forma particular de tratar os problemas e de criar as respectivas soluções.

Quais as razões para estudar os conceitos de Paradigmas de Linguagens ou Paradigmas de Programação?

O papel do desenvolvedor de *software* ou programador tornou-se essencial na sociedade que está ficando cada vez mais digital. Então, por que o ensino sobre as Ciências da Computação e os Sistemas de Informação está se intensificando, agora? O que é diferente do passado?

- A diferença é que o avanço tecnológico tem sido de tal ordem que requer um número muito maior de desenvolvedores de *software* altamente treinados e capacitados. A economia, os serviços e os métodos de produção também se aprimoraram, de forma que, atualmente, existe a necessidade de se digitalizar os processos e a informação;
- A ênfase em Tecnologia da Informação surge muito mais como consequência da rapidez com que vêm acontecendo as mudanças tecnológicas, e com as novas demandas no mercado de trabalho e não, apenas, como um modismo;
    - A competição acirrada entre as empresas concorrentes e na economia também força o mercado a adotar as tecnologias diferentes para atender às necessidades dos empreendedores.

Motivos para aprender os conceitos de Linguagens de Programação

Maior capacidade para expressar as ideias e transformar as soluções computacionais em programas de computador.

Melhorar a habilidade e o potencial de um desenvolvedor para utilizar uma linguagem já conhecida.

Quanto maior o conhecimento a respeito das funcionalidades, partes antes desconhecidas e não utilizadas das linguagens e da implementação de uma LP, maior é a possibilidade do desenvolvedor construir *softwares* mais eficientes e melhores.

- O desenvolvedor tem mais embasamento para selecionar uma linguagem de programação mais apropriada para determinado projeto de *software*, por estar familiarizado com os recursos e as opções fornecidos por essa linguagem, e como implementar esses recursos em um ambiente de produção.

Motivos para aprender os conceitos de Linguagens de Programação

Maior aptidão e facilidade para aprender novas linguagens de programação.
O desenvolvedor que domina os conceitos de determinado paradigma tem mais habilidade para aprender as linguagens que dão suporte àquele paradigma.

O desenvolvimento de *software* ainda é uma área de conhecimento relativamente nova.
As metodologias de gerenciamento de projetos, modelagem de *software* e programação estão em constante evolução.

O conhecimento prévio do paradigma de uma LP pode diminuir a sua curva de aprendizado.
Em linguagem natural, quanto mais conhecimento se tem sobre a gramática de seu idioma nativo, mais facilidade terá para aprender uma segunda língua (SEBESTA, 2011).

Motivos para aprender os conceitos de Linguagens de Programação

Um desenvolvedor com um aprendizado mais amplo dos recursos da LP torna-se menos limitado na programação de computadores.
Quanto maior a habilidade e entendimento na implementação e nas funcionalidades de uma LP, mais possibilidades um desenvolvedor tem para desenvolver os *softwares* mais eficientes e melhores.

O desenvolvedor tem maiores possibilidades e conhecimento para criar as linguagens de programação.

Conceitos básicos e fundamentais sobre as Linguagens de Programação

- Um programa também é um *software*, ou seja, um programa de computador que possibilita que novos programas sejam escritos, facilitando para que o desenvolvedor de *software* elabore os seus programas.
- Um ambiente de desenvolvimento integrado (IDE) reúne as ferramentas essenciais para o desenvolvimento e o teste de *software* em um único ambiente gráfico.
- Uma IDE utiliza um compilador que, também, é um programa que converte o código-fonte gerado em uma linguagem de programação de alto nível para um código de máquina de baixo nível.
    - Quando trabalhamos com o desenvolvimento de *software*, precisamos entender que, além da LP, também precisamos nos preocupar tanto com o sistema operacional (SO), quanto com o *hardware* onde será executada essa LP.
    - Também existem algumas linguagens interpretadas, como Java e Python, que são mais independentes do sistema operacional.

Conceitos básicos e fundamentais sobre as Linguagens de Programação

- Cada linguagem de programação pode dar suporte a um ou mais paradigmas de programação.
- As LPs seguem os padrões de codificação binária, com as semânticas e a sintaxe próprias.
- O desenvolvedor de *software* se utiliza dessas bibliotecas de conjuntos de funções, códigos, e recursos padrões e nativos para criar sistemas que resolvem os problemas das organizações e pessoas.
- Após concluído o código-fonte e as regras verificadas, o compilador traduz o programa que foi escrito para o computador em binário, para que seja executado.
    - Em relação à forma de tradução do código-fonte para o código de máquina ou binário, as LPs podem ser classificadas em **interpretadas ou compiladas**.
    - Também existem as abordagens híbridas, como a do Java, que é compilada, mas requer uma máquina virtual para a execução do *bytecode* (SEBESTA, 2011).

Conceitos básicos e fundamentais sobre as Linguagens de Programação

- As linguagens de baixo nível utilizam a linguagem de máquina como as instruções a serem executadas pelo processador. A CPU executa, nativamente, a instrução em linguagem de máquina (ou binária), que consiste em uma sequência de *bytes*, onde cada *byte* significa algo para o processador.

- Por exemplo, a seguinte frase: “Estudo para ter mais conhecimento” está representada a seguir, de acordo com a tabela ASCII e em binário:

01000101 01110011 01110100 01110101 01100100 01101111
00100000 01110000 01100001 01110010 01100001 00100000
01110100 01100101 01110010 00100000 01101101 01100001
01101001 01110011 00100000 01100011 01101111 01101110
01101000 01100101 01100011 01101001 01101101 01100101
01101110 01110100 01101111

Interatividade

O programa que utiliza a CPU para traduzir um programa-fonte, um comando de cada vez, de uma linguagem de programação interpretada, geralmente, de alto nível e o converte em um código executável é denominado:

a) Compilador.
b) Interpretador.
c) Editor de texto.
d) Depurador.
e) Tradutor.

Resposta

O programa que utiliza a CPU para traduzir um programa-fonte, um comando de cada vez, de uma linguagem de programação interpretada, geralmente, de alto nível e o converte em um código executável é denominado:

a) Compilador.
b) Interpretador.
c) Editor de texto.
d) Depurador.
e) Tradutor.

Processo de desenvolvimento de programas

- Todo programa ou código-fonte, não importa em qual linguagem de programação seja desenvolvido, antes de ser executado, tem que ser convertido para a linguagem de máquina, por meio de 3 métodos: compilação ou tradução, interpretação pura e sistema híbrido (SEBESTA, 2011).
- A linguagem de máquina, gerada a partir do código-fonte, terá que ser compatível com o *hardware* e o sistema operacional onde esse programa será executado.

Compilação ou tradução:

- Na compilação ou tradução, a conversão do programa-fonte escrito em uma linguagem de alto nível é traduzida para o código executável, em uma versão compatível com a linguagem de máquina, a qual pode ser executada, diretamente, no computador;
- As LPs Ada, C e COBOL utilizam o processo de compilação.

Processo de desenvolvimento de programas

A linguagem que um compilador traduz é chamada de programa-fonte. As fases mais importantes do processo de compilação são as seguintes:

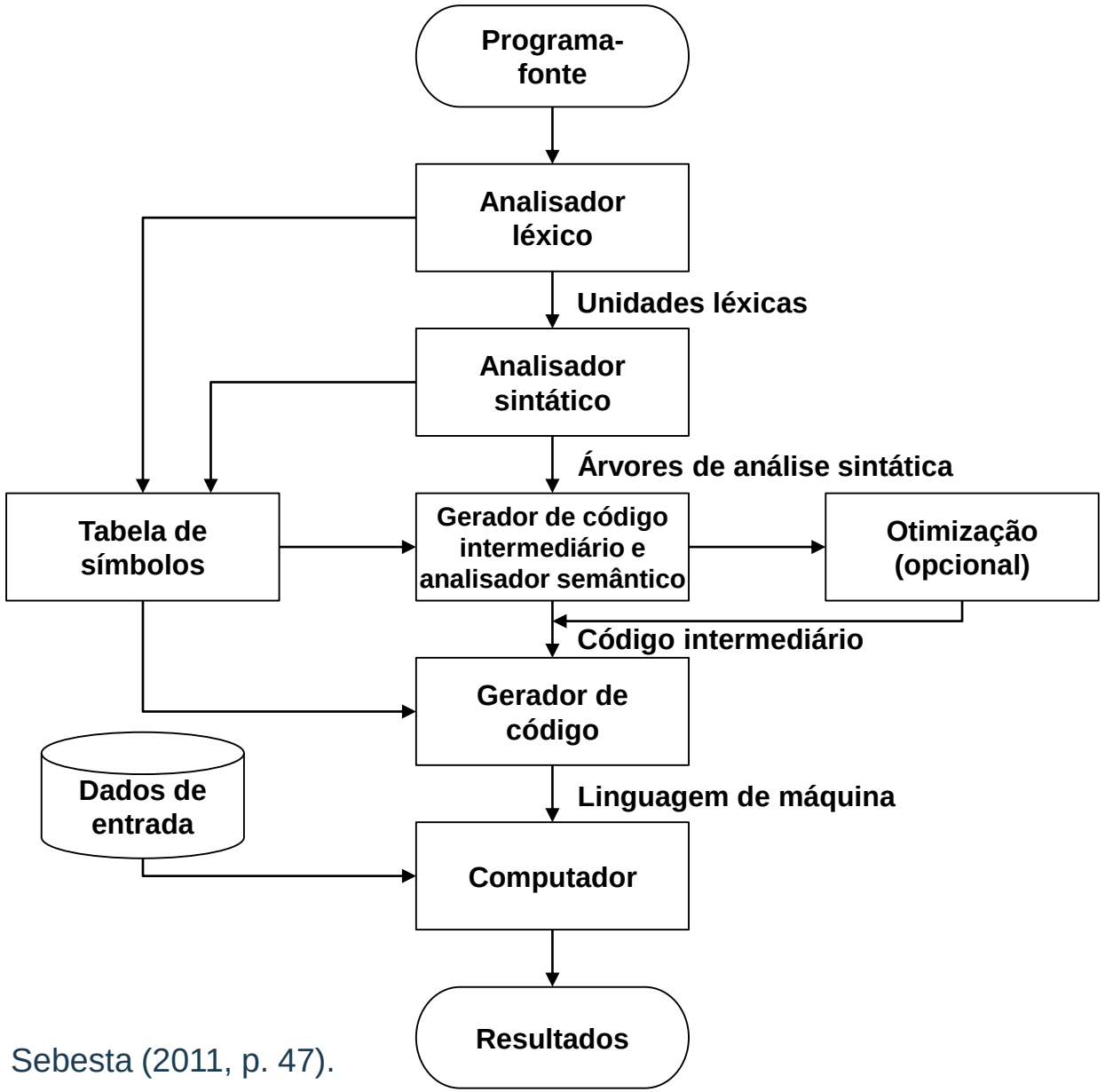
- **Analisador léxico:** o analisador léxico agrupa os caracteres do programa-fonte em unidades léxicas ou *tokens*, que são: identificadores, símbolos, operadores e palavras especiais, ignorando os comentários no programa-fonte;
- **Analisador sintático:** esse analisador aplica as regras gramaticais da linguagem sintática e confere, a partir da sequência de *tokens*, se estes compõem as estruturas sintáticas, como os comandos e as expressões;
- **Gerador de código intermediário e analisador semântico:** o gerador de código intermediário cria um programa em uma linguagem diferente, em um nível intermediário entre o programa-fonte e a saída final do compilador;
  - O analisador semântico é parte do gerador de código intermediário, que verifica os erros de tipos em comandos e expressões;
  - Analisa se as estruturas sintáticas fazem sentido, como o escopo e o uso dos nomes, e a correspondência entre as declarações e o uso de variáveis, e se há uma relação entre os operadores e os operandos.

Processo de desenvolvimento de programas

- Para o processo de compilação, a tabela de símbolos serve como uma base de dados onde estão armazenadas as informações de tipo e os atributos de cada um dos nomes definidos pelo usuário no programa.
- Essa informação é inserida na tabela pelos analisadores sintático e léxico, e usada pelo analisador semântico e pelo gerador de código.

Processo de desenvolvimento de programas

As fases mais importantes do processo de compilação são exibidas na figura ao lado (SEBESTA, 2011):



Fonte: Adaptado de: Sebesta (2011, p. 47).

Processo de desenvolvimento de programas

Interpretação:

- Esse processo de conversão do código-fonte utiliza a CPU para traduzir um programa-fonte, de uma LP interpretada, geralmente, de alto nível e executa, diretamente, o programa;
- Esse processo de *software* fornece uma máquina virtual para a linguagem (SEBESTA, 2011).

As desvantagens desse processo, em relação à compilação, são:

- Tempo de execução, que é de 10 a 100 vezes mais lento devido ao processo de decodificação das sentenças em linguagem de máquina;
- Maior consumo de espaço em memória de máquina.

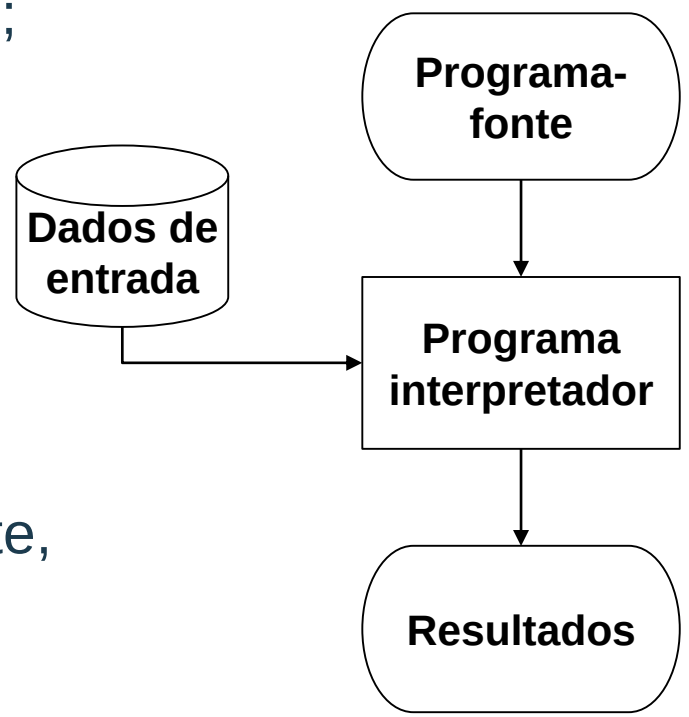
- Nos anos 1960, as LPs LIPS, SNOBOL e APL eram, puramente, interpretadas;
- Com o desenvolvimento da *web*, as linguagens de *scripting*, como PHP e JavaScript, passaram a utilizar o processo de interpretação (SEBESTA, 2011).

Processo de desenvolvimento de programas

Interpretação pura:

As principais vantagens do interpretador, em relação aos programas compilados, são:

- Simplicidade: os programas são menos complexos e menores do que os dos compiladores;
- Portabilidade: o código pode ser aceito em qualquer plataforma que possua um interpretador;
- Confiabilidade: devido ao fato de terem acesso direto ao código do programa, podem oferecer, ao desenvolvedor, as mensagens de erro mais assertivas;
  - Processo de interpretação é similar ao de tradução quando duas pessoas conversam em idiomas diferentes, por exemplo, português e espanhol, e não têm o domínio do idioma;
  - Então, elas precisam de um intérprete, que é o responsável por fazer a tradução ou a conversão de um idioma para o outro.



Fonte: Adaptado de: Sebesta (2011, p. 49).

Processo de desenvolvimento de programas

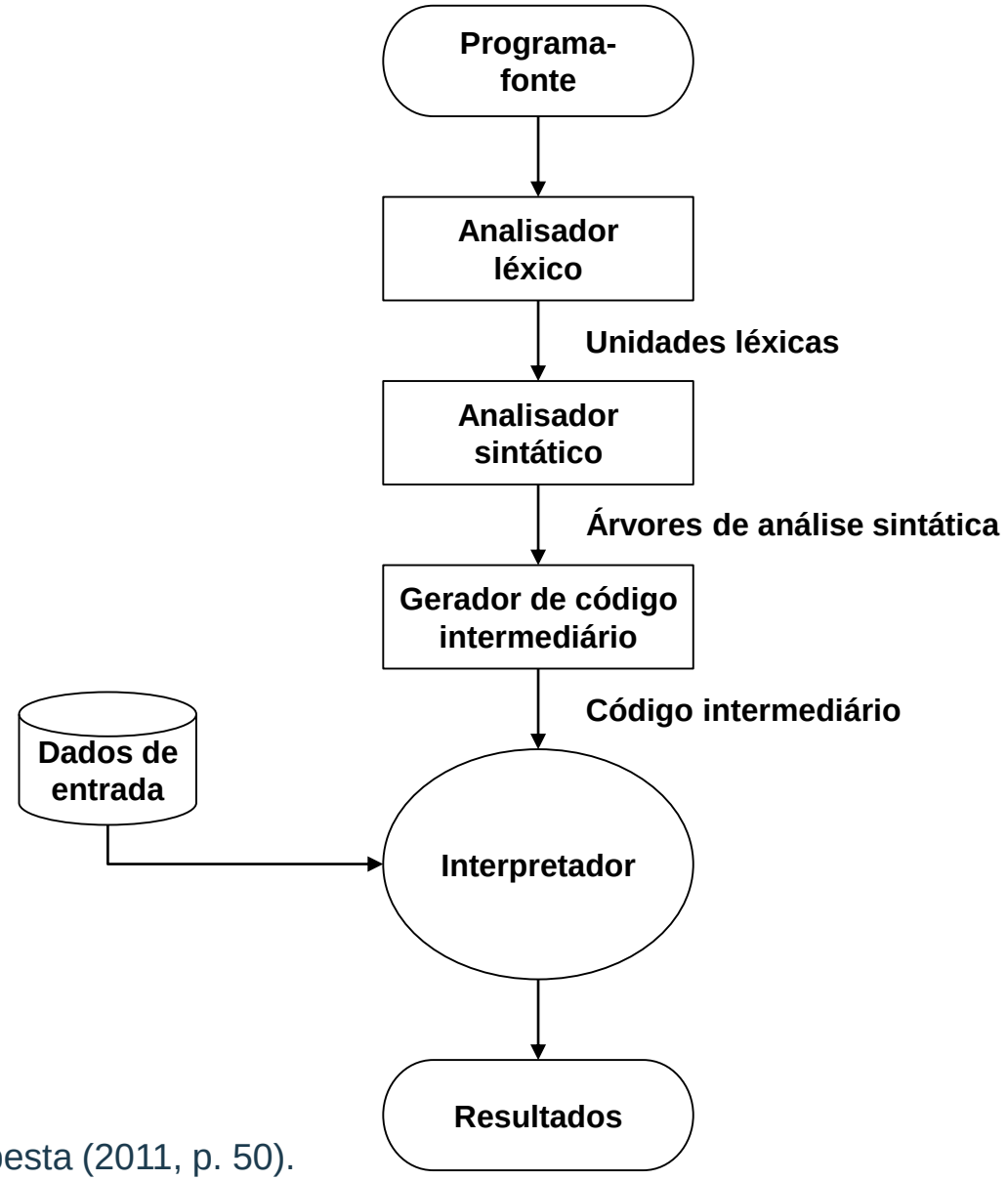
Interpretação híbrida:

- A interpretação híbrida é um meio termo entre os compiladores e os interpretadores puros; ela realiza a tradução de linguagens de alto nível para uma linguagem intermediária projetada para facilitar a interpretação;
- Esses sistemas são mais rápidos do que a interpretação pura, porque as sentenças da linguagem-fonte são decodificadas apenas uma vez, e o programa interpreta o código intermediário, ao invés de traduzir o código da linguagem intermediária para a linguagem de máquina (SEBESTA, 2011).

Processo de desenvolvimento de programas

Interpretação híbrida:

- A interpretação híbrida incorpora o melhor de cada sistema: interpretação, tradução e compilação, combinando a execução rápida dos compiladores com a portabilidade dos interpretadores, gerando um código intermediário que pode ser implementado em qualquer sistema operacional: Windows, Linux, Unix e MAC;
- Para cada sistema operacional existe um interpretador apropriado.

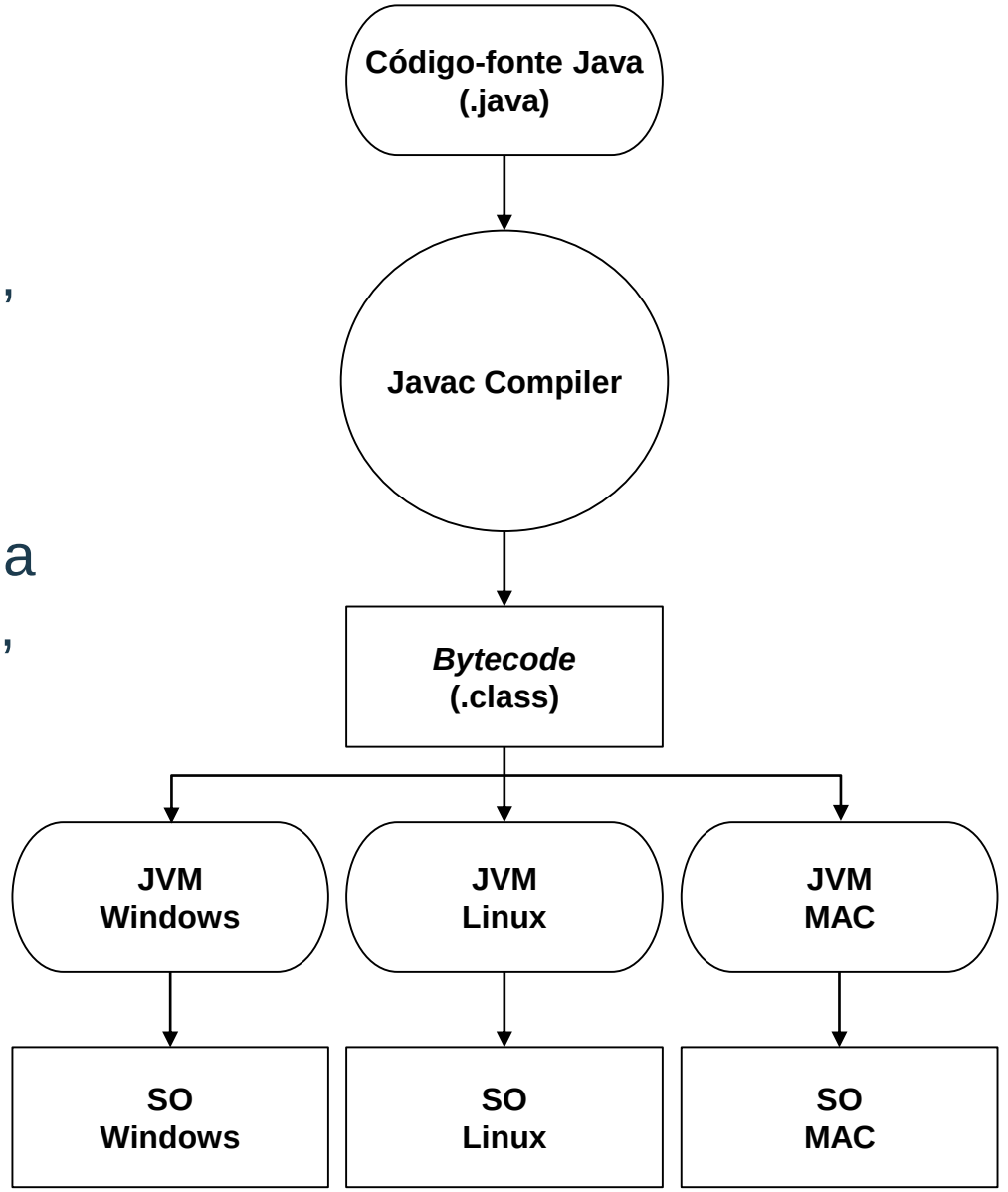


Fonte: Adaptado de: Sebesta (2011, p. 50).

Processo de desenvolvimento de programas

Java: exemplo de interpretação híbrida:

- Desenvolvida pela Sun Microsystems, a LP Java é a linguagem híbrida mais conhecida, na atualidade;
- Essa LP requer a compilação em *bytecode* que, depois, deve ser interpretada. Tornou-se famosa pelo seu uso no ambiente *web*. Seu código-fonte, com a extensão .java, é traduzido na forma de *bytecode*, um código intermediário, que pode ser executado por uma máquina virtual Java (JVM) em diversos ambientes operacionais, como: Windows, Linux, Unix e MAC.



Fonte: Adaptado de: Sebesta (2011).

Processo de desenvolvimento de programas

Comparação entre a compilação e a interpretação:

Tradução ou Compilação	Interpretador puro	Interpretador híbrido
Gera um código executável.	Sem a geração de um código.	Geração de código intermediário.
Execução rápida. Tradução lenta.	Execução lenta. Independente de plataforma.	Execução não muito rápida. Tradução rápida.
Depende da plataforma de execução.	Independe da plataforma de execução.	

Fonte: Autoria própria.

Interatividade

Quando o programa-fonte, escrito em uma linguagem de alto nível, é traduzido para o código executável, em uma versão compatível com a linguagem de máquina, a qual pode ser executada diretamente no computador, recebe o nome de:

a) Extensão.
b) Interpretação.
c) Edição de texto.
d) Depuração.
e) Compilação ou tradução.

Resposta

Quando o programa-fonte, escrito em uma linguagem de alto nível, é traduzido para o código executável, em uma versão compatível com a linguagem de máquina, a qual pode ser executada diretamente no computador, recebe o nome de:

a) Extensão.
b) Interpretação.
c) Edição de texto.
d) Depuração.
e) Compilação ou tradução.

Estilo e qualidade de programas

Quais as propriedades desejáveis de uma Linguagem de Programação?

- **Legibilidade:** é a facilidade com que um código-fonte pode ser lido e entendido;
- Programas difíceis de ler são, também, difíceis de escrever e modificar, prejudicando a confiabilidade na LP, tanto nas fases de desenvolvimento quanto nas de manutenção do ciclo de vida (SEBESTA, 2011);
- Quanto mais a linguagem de uma LP for natural, a sua leitura se tornará menos complicada e mais fácil será o seu entendimento.

- **Redigibilidade e simplicidade:** uma LP deve propiciar ao desenvolvedor a facilidade de se concentrar nos algoritmos centrais de um programa para a resolução do problema de negócio, e não se preocupar com os aspectos não relevantes, como, por exemplo, os detalhes de implementação, ou seja, o programador ter que se preocupar como é que o programa vai se comportar no *hardware*.

Estilo e qualidade de programas

Quais as propriedades desejáveis de uma Linguagem de Programação?

- **Facilidade de aprendizado:** uma LP deve fornecer o suporte para que o profissional de TI, que trabalha desenvolvendo *softwares*, para que os computadores tenham condições de assimilar, rapidamente, os conceitos e a utilização dessa LP;
- Como, por exemplo, o tempo da curva de aprendizado entre as LPs Assembler e Java.

- **Eficiência:** a LP deve fornecer os meios adequados para que determinado tipo de aplicação possa atingir os seus objetivos, por exemplo, a linguagem Python fornece o suporte para as aplicações de Inteligência Artificial e da Ciência de Dados.

Estilo e qualidade de programas

Quais as propriedades desejáveis de uma Linguagem de Programação?

- **Reusabilidade:** possibilidade que um *software* tem de poder ser reutilizado ao todo, ou em parte, por outras aplicações, aumentando a sua flexibilidade e portabilidade, e reduzindo os custos de desenvolvimento e de manutenção.

- **Portabilidade:** a portabilidade é influenciada pelo grau de padronização da LP; é a capacidade de um programa de se comportar de maneira similar, independente da arquitetura computacional, do *hardware* ou do sistema operacional sobre os quais está sendo executado;
  - Um exemplo seria um programa que pudesse ser executado tanto em uma máquina Linux quanto em uma máquina Windows.

Estilo e qualidade de programas

Quais as propriedades desejáveis de uma Linguagem de Programação?

O custo de uma linguagem é medido em razão de várias de suas características:

- Tanto o custo de escrever os programas quanto o custo de treinar os programadores pode ser reduzido quando um ambiente de programação é bem otimizado e padronizado;
- O custo de escrita da linguagem depende da facilidade de escrita da linguagem;
- O custo de compilar os programas em uma linguagem de programação;
- O custo de implementação da linguagem influencia na utilização da linguagem (ex.: Java);
- A confiabilidade em uma LP também é um custo em caso de falha do sistema;
    - O custo de manutenção de *software* depende de uma série de características da linguagem, sobretudo, da legibilidade, devido ao fato de, geralmente, a manutenção ser feita por desenvolvedores diferentes daqueles que desenvolveram o sistema.

Estilo e qualidade de programas

Quais as propriedades desejáveis de uma Linguagem de Programação?

- **Tratamento de exceção:** evidente que uma LP será mais confiável se conseguir interceptar os erros em tempo de execução, promover, facilmente, as medidas corretivas e permitir que o programa continue sem o interrompimento, devido à ocorrência de um erro, tornando o comportamento de um programa mais previsível;
- Alguns exemplos de LPs que implementam o tratamento de exceções são: C++, C# e Java (SEBESTA, 2011).

Interatividade

Avalie as assertivas sobre as propriedades desejáveis de uma LP:

I. Eficiência: é a capacidade da LP de se adaptar a qualquer sistema operacional.

II. Legibilidade: é a facilidade com que um código-fonte pode ser lido e entendido.

III. Facilidade de aprendizado: uma LP deve fornecer o suporte para que o profissional de TI tenha as condições de assimilar, rapidamente, os conceitos e a utilização dessa LP.

IV. Portabilidade: uma LP deve propiciar ao desenvolvedor a facilidade de se concentrar nos algoritmos centrais de um programa, e não se preocupar com os aspectos não relevantes, como, por exemplo, os detalhes de implementação.

Após a sua avaliação, assinale a opção que apresenta a(s) assertiva(s) correta(s):

a) I, II, III e IV.

b) II e III.

c) III.

d) II e IV.

e) IV.

Resposta

Avalie as assertivas sobre as propriedades desejáveis de uma LP:

I. Eficiência: é a capacidade da LP de se adaptar a qualquer sistema operacional.

II. Legibilidade: é a facilidade com que um código-fonte pode ser lido e entendido.

III. Facilidade de aprendizado: uma LP deve fornecer o suporte para que o profissional de TI tenha as condições de assimilar, rapidamente, os conceitos e a utilização dessa LP.

IV. Portabilidade: uma LP deve propiciar ao desenvolvedor a facilidade de se concentrar nos algoritmos centrais de um programa, e não se preocupar com os aspectos não relevantes, como, por exemplo, os detalhes de implementação.

Após a sua avaliação, assinale a opção que apresenta a(s) assertiva(s) correta(s):

a) I, II, III e IV.

b) II e III.

c) III.

d) II e IV.

e) IV.

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

- Escopo é a característica que determina um local onde uma variável pode ser utilizada ou referenciada como um identificador em um programa;
- Uma variável declarada dentro de um procedimento é, normalmente, local; o contexto define o escopo;
- De maneira geral, um escopo pode ter uma visibilidade estática, feita antes da execução do programa, e não muda a dinâmica que se transforma durante a execução do programa;
- No escopo estático, as variáveis são associadas ou amarradas em tempo de compilação do programa, como, por exemplo: uma variável e o valor atribuído a ela;
    - A maioria das LPs imperativas utilizam o escopo estático, por causa dos subprogramas, criando, assim, uma hierarquia de escopo;
    - Os blocos de códigos (função ou módulo) são conjuntos de instruções executados em sequência, delimitados por marcadores dependendo da LP.

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

Exemplos de delimitadores de blocos:

Linguagem	Delimitador de bloco
Ada, Algol e Pascal	Begin/End
Java e C#	{} (Chaves)
Python	Indentação e espaços em branco

Fonte: Autoria própria.

- Nota: o Python, ao contrário de outras linguagens, utiliza os espaços em branco e a indentação para separar os blocos de código.

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

- Exemplo de delimitadores de bloco em Java;
- Observe que, nas linhas 1 e 5, as chaves definem o escopo da classe. Nas linhas 2 e 4, as chaves definem o escopo do método principal, o *main*.

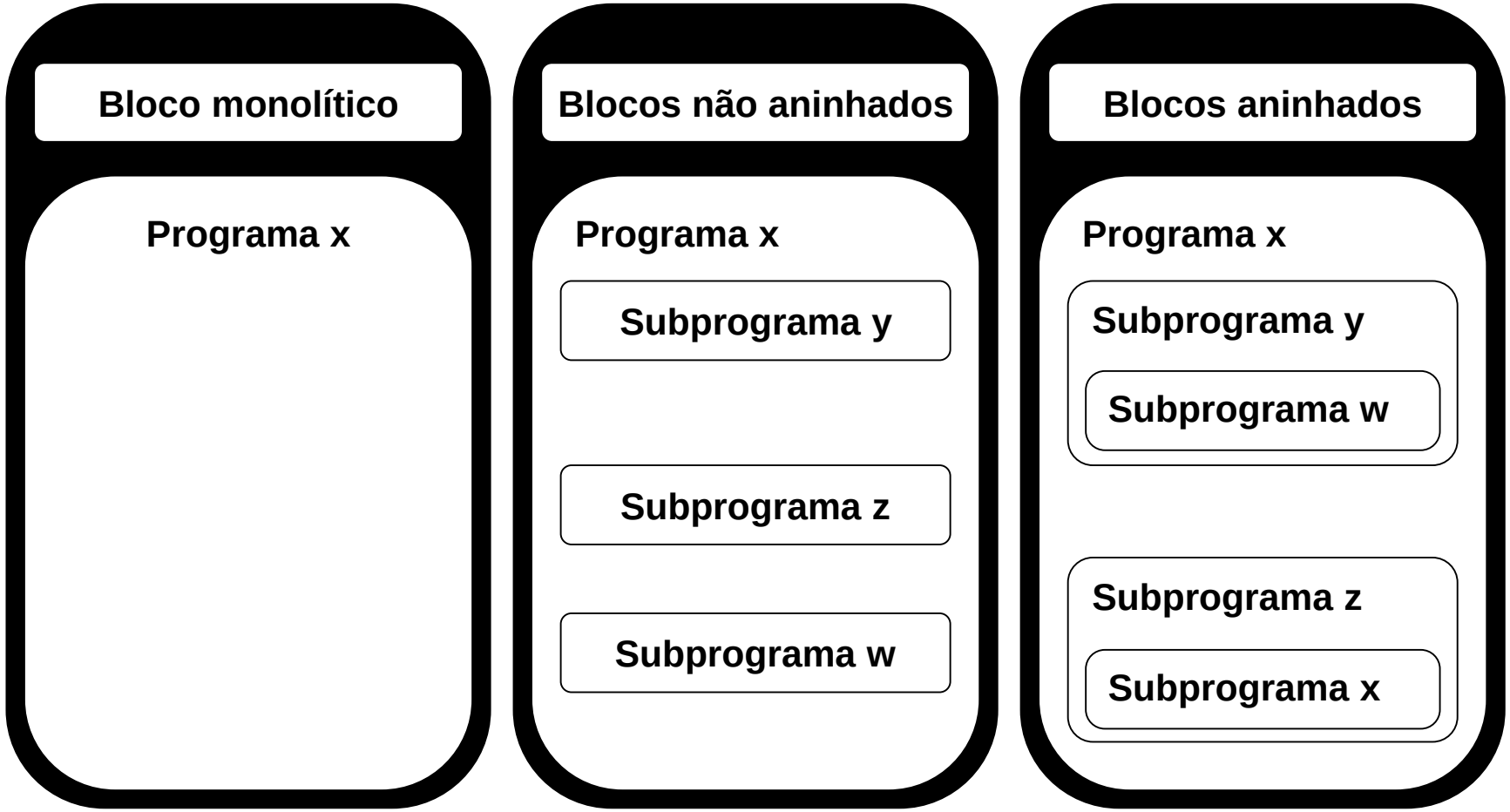
Linha	Instrução
L1	public class DelimitadoresDeBlocos {
L2	public static void main(String args[]) {
L3	System.out.println(“As chaves são delimitadores de blocos na LP Java e definem o escopo”);
L4	}
L5	}

Fonte: Autoria própria.

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

Os blocos de códigos podem ser estruturados de 3 formas: monolítico, blocos aninhados e blocos não aninhados, conforme a figura a seguir (VAREJÃO, 2004):

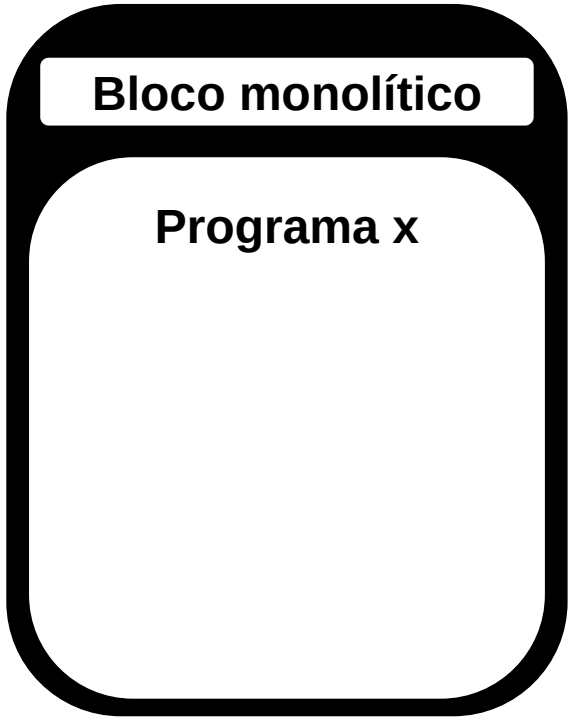


Fonte: Adaptado de: Varejão (2004).

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

- No bloco monolítico, comum nas versões antigas do Basic e Cobol, o programa todo é codificado em um único bloco;
- O escopo de visibilidade das amarrações é para o programa todo;
- Essa estrutura torna-se complicada quando o programa começa a ficar muito grande, pois não tem a modularização que divide o código em vários pedaços interligados.

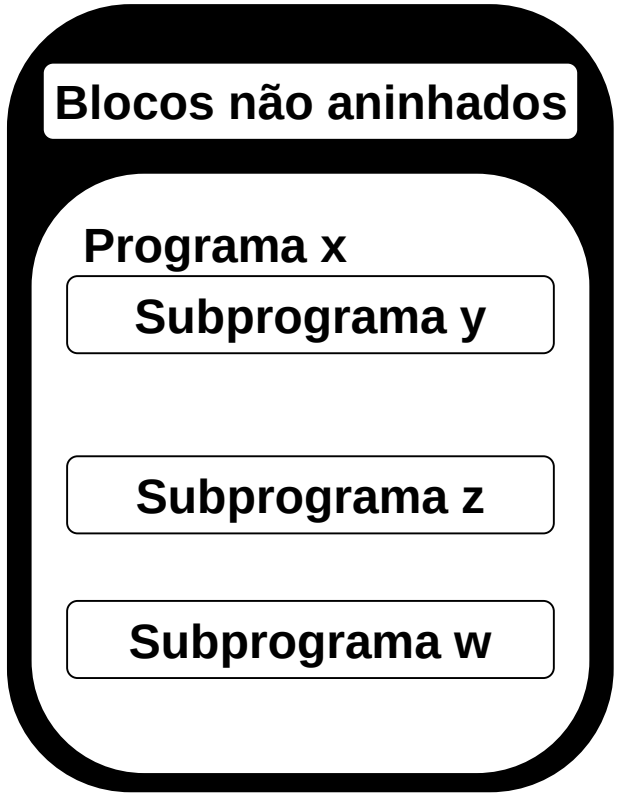


Fonte: Adaptado de: Varejão (2004).

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

- Na divisão de blocos não aninhados, o programa é subdividido em subprogramas; no exemplo da figura a seguir, temos os subprogramas y, z e w;
- O escopo de visibilidade das amarrações é definido pelo bloco. Onde os identificadores são criados, são chamados de locais, se declarados dentro do bloco; e globais, se declarados fora do bloco;
- A LP Fortran utiliza esses blocos não aninhados.

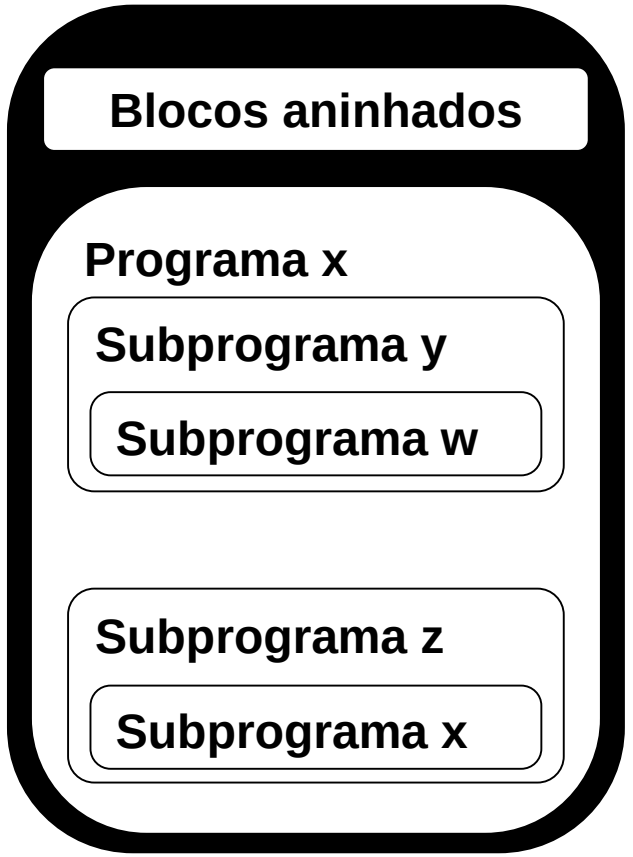


Fonte: Adaptado de: Varejão (2004).

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

- A estrutura de blocos aninhados é mais avançada, porque qualquer bloco pode ser criado dentro de outro bloco e, assim, sucessivamente;
- Nesse caso, a visibilidade é do escopo ou do bloco mais interno, o nível de bloco onde está sendo utilizado, para o bloco mais externo ou superior;
- Essa estrutura é utilizada pelas LPs Java, C#, C++, C e Pascal.



Fonte: Adaptado de: Varejão (2004).

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

Consta, a seguir, um exemplo de bloco aninhado em Java:

```
public class ProgramaX {
    String variavelProgramaX = “ProgramaX”;
    public static void main(String[] argumentos) {
            ProgramaX programaX = new ProgramaX();
            programaX.subProgramaY();
    }
                                        public void subProgramaY() {
                                                String variavelSubProgramaY = “SubProgramaY”;
                                                System.out.println(variavelSubProgramaY + “ executando.
                                        De dentro do “ + variavelProgramaX).
```

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

- No escopo dinâmico, a amarração muda durante a execução do programa, ou seja, quando os procedimentos ou os subprogramas vão sendo executados, ou conforme o fluxo de controle do programa;
- Exemplo de LPs que implementam esse escopo são: Lisp, Perl e SNOBOL4. O tipo de escopo (dinâmico ou estático) depende da linguagem. Nesse caso, podem ocorrer os problemas de eficiência do programa devido ao fato que, durante a execução, tem de ser realizada a checagem dos tipos de dados. Também podem ocorrer os problemas de legibilidade, ou seja, o programador tem que entender a amarração que foi realizada e os problemas de confiabilidade, pois o subprograma pode acessar as variáveis locais do bloco que fez a chamada.

Estilo e qualidade de programas

Definição do escopo de visibilidade das variáveis:

Escopo dinâmico, exemplo em pseudocódigo:

```
procedimento procedimentoPrincipal() {
    string informeLetra = “A”;
procedimento subProcedimento1() {
    escreva(informeLetra);
}
procedimento subProcedimento2() {
    string informeLetra = “B”;
    subProcedimento1();
}
    subProcedimento2();
    subProcedimento1();
}
```

Interatividade

É uma característica que determina um local onde uma variável pode ser utilizada ou referenciada como um identificador em um programa. Estamos definindo o(a)s:

a) Extensão.
b) Delimitadores de bloco.
c) Bloco monolítico.
d) Escopo.
e) Tempo de compilação.

Resposta

É uma característica que determina um local onde uma variável pode ser utilizada ou referenciada como um identificador em um programa. Estamos definindo o(a)s:

a) Extensão.
b) Delimitadores de bloco.
c) Bloco monolítico.
d) Escopo.
e) Tempo de compilação.

Referências

- DAURICIO, J. S. *Algoritmos e lógica de programação*. Belo Horizonte: Editora e Distribuidora Educacional S. A., 2015.
- ETZION, O.; NIBLETT, P. *Event processing in action*. Stamford: Manning Publications Co., 2011.
- FURGERI, S. *Java 8*: ensino didático – desenvolvimento e implementação de aplicações. São Paulo: Érica, 2015.
- SEBESTA, R. W. *Conceitos de linguagens de programação*. 9. ed. Porto Alegre: Bookman, 2011.
- TUCKER, A. B.; NOONAN, R. E. *Linguagens de programação*: princípios e paradigmas. 2. ed. São Paulo: McGraw Hill, 2009.
- VAREJÃO, F. M. *Linguagens de programação*: conceitos e técnicas. Rio de Janeiro: Elsevier (Campus), 2004.

ATÉ A PRÓXIMA!