



UNIVERSIDADE FEDERAL DO CEARÁ – CAMPUS SOBRAL
CURSO DE ENGENHARIA DE COMPUTAÇÃO
DISCIPLINA: Paradigmas e Linguagem de Programação
PROFESSOR: Iális Cavalcante de Paula Junior

LISTA 1 – Avaliação de uma linguagem de programação

ALUNOS	MATRÍCULA
Francisco Cassiano de Vasconcelos Souza	413067
Francisco Renato Graciano Freire	428131
Francisco Jefferson Marques de Sousa	475486
Francisco Evandro Ribeiro Martins Filho	385143

Sobral – CE

2021

SUMÁRIO

1. HISTÓRICO	3
1.1 Clipper.....	
1.2 Haskell.....	
1.3 Java.....	
2. PARADIGMAS	4
2.1 Clipper.....	
2.2 Haskell.....	
2.3 Java.....	
3. INTRODUÇÃO À SINTAXE	5
3.1 Clipper.....	
3.2 Haskell.....	
3.3 Java.....	
4. FORMAS DE VINCULAÇÕES DE TIPOS	5
4.1 Clipper.....	
4.2 Haskell.....	
4.3 Java.....	
5. CÓDIGO COMENTADO	6
5.1 Clipper.....	
5.2 Haskell.....	
5.3 Java.....	
6. REFERÊNCIAS BIBLIOGRÁFICAS	8

1. HISTÓRICO

1.1 Clipper

Ao analisar e discutir a ideia de criar um compilador e fundar uma empresa para comercializá-lo, dois amigos resolvem criar com nome de Clipper, o nome veio de um quadro no restaurante que frequentavam. Com nome Nantucket Clipper, vinha com propósito de ser o melhor compilador para dBase, popular gerenciador de banco de dados da época, que existia.

Entre os anos de 1985 até 1987 Nantucket Corporation teve algumas versões comercializadas. Na década de 1990 lança-se o Clipper 5, com essa versão Clipper inicia sua desvinculação com a dBase tornando-se uma linguagem de programação com evolução própria. Embora hoje seja considerada uma linguagem obsoleta dado que parou de evoluir após a versão 5.3, Clipper ainda possui uma razoável base de programadores conhecidos pelo depreciativo apelido de "clippeiros".

1.2 Java

A linguagem de programação Java teve seu início no ano de 1991, pela empresa Sun Microsystems(hoje pertencente à empresa Oracle). Fomentada por um projeto desenvolvido por três engenheiros Patrick Naughton, Mike Sheridan, e James Gosling, intitulado Projeto Green.

Com intenção de revolucionar o mundo da tecnologia moderna de softwares e de empresas eletrônicas de consumo, veio a criação da linguagem que inicialmente chamava-se Oak, nome dado por James Gosling referente a uma árvore, mais tarde chamada pelo nome popular, Java nome da terra de origem do café que os desenvolvedores apreciavam.

Em 1995 adaptaram a linguagem para a internet e foi um sucesso, pois a web na época não tinha interatividade, estabelecendo uma aceitação aos navegadores populares da época. A tecnologia Java expandiu rapidamente sua utilização e empresas gigantes adotaram o uso em seus produtos como a IBM. Dentre várias características positivas da linguagem Java, o principal item é o fato de ter uma “Independência de plataforma”.

1.3 Haskell

Haskell é uma linguagem puramente funcional, batizada com esse nome em homenagem ao matemático Haskell Brooks Curry, que teve contribuições relevantes para a programação funcional. É uma linguagem relativamente nova, derivada de outras linguagens funcionais como Lisp, Scheme, LM e outras. Muito utilizada no meio acadêmico, focada no alcance de solução para problemas matemáticos.

Em 1988 em reunião do comitê, métricas da nova linguagem foram discutidas, deveria ter uma sintaxe e semântica formal, de fácil ensino e assim nasceu Haskell que unia as linguagens do mesmo paradigma em uma.

As primeiras versões da linguagem foram Haskell 1.0 até 1.4 na década de 1990, após 7 anos foi lançada a versão Haskell 98 que sofreu modificações para evolução da linguagem e em 2009 lançou-se a versão Haskell 2010, com melhoramento e atualizações incremental na linguagem em diversos aspectos.

2. PARADIGMAS

2.1 Clipper

Clipper é pertencente ao Paradigma procedural, nesse tipo de construção, as instruções devem ser passadas ao computador na sequência em que devem ser executadas, (como Pascal, C, Ada, Cobol, Fortran, Clipper). Linguagens procedurais são aquelas no qual nosso código é dividido em sub-rotinas (procedures) ou function (funções). Uma procedure é uma função sem retorno (pode ser visto como uma função que não retorna nada, void). O Clipper trata-se de uma linguagem compilada, pois foi desenvolvida para uso primário de software.

Principais aplicações;

A linguagem clipper é fortemente usada em aplicações comerciais, no desenvolvimento de sistemas de bancos de dados com designer gráfico

2.2 Java

Java pertence e foi precursor do paradigma orientado a objeto na década de 90, objetivando permitir uma programação multiplataforma de uma mesma maneira. O paradigma orientado a objeto buscou apoiar-se nas características de classe e objeto ao tentar retratar a programação tal qual se enxerga o mundo real.

Nesse paradigma, todos os objetos têm determinados estados e comportamentos. Esses estados são descritos pelas classes como atributos. Já a forma como eles se comportam é definida por meio de métodos, que são equivalentes às funções do paradigma funcional. Para que uma linguagem de programação seja do tipo de paradigma orientado a objetos, deve implementar seus três alicerces básicos, que são conceito de herança, polimorfismo e encapsulamento.

Diferente das linguagens de programação modernas, a linguagem Java é compilada para um bytecode e interpretada para uma máquina virtual (JVM), portanto, tratando-se de uma linguagem híbrida.

Java é uma das linguagens de programação mais populares usadas para criar aplicações e plataformas Web. Foi desenvolvido para ter flexibilidade, permitindo que os códigos seriam executados em qualquer máquina, independentemente da arquitetura ou plataforma. Há muitos ambientes em que o Java é aplicado no mundo real, desde aplicações de smartphones, aplicações científicas e financeiras, e está presente também no mundo dos gamers como no jogo Minecraft.

Sendo assim, o Java está presente em uma gama de aplicações existente do mundo real como: Aplicações Android, Aplicações na indústria de serviços financeiros, Aplicativos de Web, Ferramentas de Software, Aplicação de negociação, Tecnologia de Big Data, Aplicações Científicas entre outras.

2.3 Haskell

Paradigma funcional enfatiza o processo de identificar blocos e partes repetidas de código e constroem funções que encapsulam a funcionalidade dentro de uma simples definição. O paradigma de programação funcional enxerga todos os subprogramas como funções que recebem argumentos e retornam soluções simples. A solução retornada é baseada inteiramente na entrada e o tempo em que uma função é chamada é irrelevante sendo possível a passagem de uma função como parâmetro. O programa passa a ser uma lista de funções, sendo o próprio programa principal uma lista. É um paradigma de programação que trata a computação como uma avaliação de funções matemáticas.

Haskell é uma linguagem de programação híbrida, pois tanto pode ser compilada quanto interpretada, os programas são geralmente chamados de scripts.

Alguns exemplos de suas principais aplicações: o Facebook utiliza Haskell em seus programas anti-spam, como um software de código aberto, e também em data centers. Também tem o Swift Navigation, um fabricante de GPS de alta precisão, que implementa porções significativas de seus produtos em Haskell, gerando alguns softwares de código aberto. E a Cryptol, uma linguagem e ferramenta para desenvolver e verificar algoritmos de criptografia, que é implementada em Haskell.

3. INTRODUÇÃO À SINTAXE

3.1 Clipper

Para programar em Clipper, precisa-se de um editor de textos, semelhante ao Edit do MS-DOS para escrever os programas. Os mais conhecidos são o Qedit e o Side-Kick pela sua gama de opções na editoração de textos que facilita na hora de programar.

Os programas são compostos de arquivos de texto e tem por padrão a extensão de arquivo *.PRG.

// Tudo que vem depois das duas barras é ignorado, apenas comentário.

```
CLS // LIMPA A TELA
cNOME := SPACE(35) // DEFINE VARIÁVEL cNOME C/ 35 CARACTERES EM
BRANCO
@ 01,10 SAY "QUAL É O SEU NOME ?" GET cNOME // Pergunta na linha 1, coluna
10.
READ // LÊ O(S) GET(S) - ESPERA ENTRADA DO TECLADO
// SE FINALIZAR COM [ENTER] ATRIBUI O VALOR À cNOME
// SE SAIR COM [ESC] cNOME PERMANECE = SPACE(35)
@ 02,10 SAY "VOCÊ DISSE QUE ERA "+cNOME // DIZ NA LINHA 2, COLUNA
10, SEU NOME
```

Cada linha do programa não necessita de indicadores de final de linha, como se verifica em algumas linguagens com o (;) "ponto e vírgula" (Pascal, Javascript, C etc.). Pelo contrário, se o último caractere de uma linha no Clipper for um "ponto e vírgula", indica que a linha não terminou e que a linha abaixo é continuação da linha de cima.

Ex:

```
IF (!EMPTY(cCDEBITO) .OR. !EMPTY(cCCREDITO));
    .OR. cCDEBITO # cCCREDITO
    EXIT
ENDIF
```

Seria o mesmo que:

```
IF (!EMPTY(cCDEBITO) .OR. !EMPTY(cCCREDITO)) .OR. cCDEBITO #
cCCREDITO
    EXIT
ENDIF
```

Todavia, para uma melhor visualização, preferiu-se "quebrar" a linha para não ter que ficar usando a barra de rolagem para a direita, logo prefere-se o uso desse artifício. Para compreender melhor o programa, é recomendado utilizar sempre a prática de indentação das linhas, que são os espaços regulares que são dados no início de cada uma delas para indicar que as informações fazem parte de uma *rotina* (conjunto de informações) específica a ser executada.

Ou seja, é necessário dar espaços regulares nas linhas que estiverem dentro de um *Loop* (DO WHILE...ENDDO, FOR...NEXT) ou que satisfaçam uma condição (IF...ENDIF, DO CASE...ENDCASE, DO WHILE <condição>...ENDDO).

Ex.:

```
IF !EMPTY(cCCREDITO)
  nTCRED -= tvCAMPO // ESTORNA VALOR ANTERIOR
  nTCRED += vCAMPO // LANCA VALOR ATUAL
  IF nACAO = 3
    IF SA->( DBSEEK(TMP->CREDITO + DTOS(TMP>DATA) ))
      SA->( LOCKREG())
      SA->CREDITO -= tvCA
      SA->CREDITO += vCAMPO
      SA->( DBUNLOCK())
      SA->( DBCOMMIT())
    ENDIF
  ENDIF
ENDIF
```

Os espaços, linhas em branco e linhas de comentários são simplesmente ignorados pelo compilador do Clipper.

Arrays

Um *Array* ou matriz é uma variável em clipper que armazena vários valores, onde tais valores são armazenados e consultados através de sua posição na matriz.

Vejamos um exemplo de uma matriz numérica de 4x3 (4 linhas e 3 colunas):

$$B_{4,3} = \begin{pmatrix} 10 & 20 & 40 \\ 15 & 35 & 42 \\ 53 & 25 & 37 \\ 12 & 91 & 33 \end{pmatrix}$$

Que poderia ser representada assim:

$$B_{4,3} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix}$$

Onde cada a_{nm} é um valor da matriz. Por exemplo, para saber o valor de a_{32} na matriz B seria encontrado o número 25. Analogamente os outros valores.

A matriz B poderia ser construída em Clipper da seguinte maneira, compare com a imagem:

Em Clipper:	Na Matemática:
<pre>// definindo a matriz: B := ARRAY(4,3) // atribuindo valores B[1][1] := 10 B[1][2] := 20 B[1][3] := 40 B[2][1] := 15 B[2][2] := 35 B[2][3] := 42 B[3][1] := 53 B[3][2] := 25 B[3][3] := 37 B[4][1] := 12 B[4][2] := 91 B[4][3] := 33</pre>	$B_{4,3} = \begin{pmatrix} 10 & 20 & 40 \\ 15 & 35 & 42 \\ 53 & 25 & 37 \\ 12 & 91 & 33 \end{pmatrix}$

Outra forma em Clipper, atribuição dinâmica de uma matriz:

Em Clipper:	Na Matemática:
<pre>// definindo a matriz: B := {} // atribuindo valores AADD(B, {10,20,40}) AADD(B, {15,35,42}) AADD(B, {53,25,37}) AADD(B, {12,91,33})</pre>	$B_{4,3} = \begin{pmatrix} 10 & 20 & 40 \\ 15 & 35 & 42 \\ 53 & 25 & 37 \\ 12 & 91 & 33 \end{pmatrix}$

Para consultar o valor de **a32** da matriz B em clipper pode-se usar:

? B[3][2]

3.2 Java

Em Java, a estrutura é organizada em pacotes e classes, como em outras linguagens possui sua lista de palavra reservada:

- abstract
- assert
- boolean
- break
- case
- byte
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- para
- goto
- if
- implements

- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

Java trabalha com pacotes para organizar melhor todas as aplicações do programa, um pacote é basicamente uma pasta com várias classes que é declarado na parte inicial do código, é essencial para funcionamento correto, é definido da seguinte forma: package NomeDoPacote, caso o pacote não seja especificado, ele será relacionado ao pacote padrão, package default, usando apenas palavras exclusivas.

As aplicações em Java são feitas por meio de classes, sua declaração é feita da seguinte forma:

modificador	class	nome{}
--------------------	--------------	---------------

Por convenção para maior facilidade de compreensão dos outros desenvolvedores Java, recomenda-se que o nome da classe siga uma regra que é baseada em usar letras maiúsculas no início de uma palavra e demais minúsculas.

Como em outras linguagens de programação, Java possui variáveis, sendo elas divididas em 4 tipos: variável de instância, variável de classe, variável local e parâmetros.

Variável de instância: variável não antecedida da palavra reservada static e que necessitam da declaração da classe na sua definição.

exemplo:

```
public class Quadrado {  
    public String cor = "Azul";  
}
```

Esse tipo é individual para cada objeto instanciado, a mudança em um não ocasiona a mudança em outros.

A variável de classe se declara da mesma maneira mas atribuindo a palavra reservada `static` a ela e com isso alterando a forma de mudança de características se há dois objetos desta classe terão o valor da variável em comum.

```
public class Quadrado {  
    public static String cor = "Azul";  
}
```

O último tipo é voltado para o que é solicitado por um método, função.

```
public static int media (double n1, double n2, double  
n3){  
  
    double média;  
  
    ...  
  
    return média;  
  
}
```

Neste caso, a variável parâmetro são `n1`, `n2` e `n3`.

Java é uma linguagem que diferencia letras minúsculas e maiúsculas, não permite espaço na nomenclatura da variável e nem serem iniciadas por números e podem ser declaradas em qualquer parte da função.

Java é estaticamente tipificada, possui 8 tipos de dados primitivos, as variáveis possuem um valor padrão de inicialização quando essa não é gerada pelo usuário após a declaração dela.

Matriz é um objeto que contém um número fixo de valores de um único tipo. São declaradas da seguinte forma:

- `int [] matriz;`
- `int matriz[];`

E inicializadas da seguinte forma:

- `matriz = new int [10];`

por convenção recomenda-se as duas primeiras declarações;

Cada valor dentro de uma matriz é chamada de elemento e pode ser acessado por um índice que vai de 0 a n

Se essa linha de código estiver faltando no escopo, o compilador irá gerar um erro:

```
ArrayDemo.java:4: Variable anArray may not have been
initialized.
```

Para designar um valor a um elemento basta chamar o nome da matriz colchetes com a posição do elemento sinal de atribuição e o valor:

```
matriz [ 2 ] = 10;
```

Há uma notação alternativa para atribuir valores aos elementos que é considerado um atalho que informa o tamanho da matriz e o valores dos elementos na própria declaração.

```
int [ ] matriz = { 100, 200, 300};
```

A cada vírgula entre as chaves é um elemento novo com o dado inserido.

Em Java também é possível criar um matriz de matrizes, uma matriz multidimensional, informando a quantidade de colunas e linhas, tendo a mesma declaração, inicialização, atribuição e acesso de uma matriz unidimensional.

```
int matrizBid [ ] [ ];
matrizBid = new int [ 2 ] [ 1 ];
```

Acima, uma matriz bidimensional inteira foi declarada e inicializada com 3 linhas e 2 colunas, pois a contagem de elemento começa a partir do 0.

Declaração atalho:

```
int matrizBid [ ] [ ] = { { 1, 2, 3}, {7, 1} };
```

É possível mostrar os valores de uma matriz usando um comando de impressão e referenciando a matriz com a posição que deseja ser mostrada:

```
System.out.println(matriz1[2]);
```

O uso de matriz é muito útil na linguagem Java, tanto que há uma classe dedicada a isso a `java.util.Arrays`. Com o uso desta classe é possível comparar duas matrizes, copiar os elementos de uma matriz para outra classificar a ordem dos elementos de uma matriz, entre outras funcionalidades dentro de uma classe nativa do Java.

Operadores em Java:

postfix : *expr*++ *expr*--

unary: ++*expr* --*expr* +*expr* -*expr* ~ !

multiplicative: * / %

additive: + -

relational: < > <= >=

equality: == !=

AND: &

OR: ||

ternary: ? :

incremento: = += -= *= /=.

3.3 Haskell

Em Haskell existem apenas funções, e todas as funções são unárias. Em outras linguagens de programação seriam funções binárias, ternárias, etc. Em Haskell são funções cujo valor de retorno são outras funções - o que se chama currying, termo derivado de Haskell Curry. Trabalha-se somente com funções, seus parâmetros e seus retornos. Além disso, a linguagem Haskell é case-sensitive.

Em Haskell não existem variáveis globais, apenas funções e variáveis locais, definidas dentro do escopo de cada função. Também não há estruturas de loop, não possui comandos de repetição como While e For, ou instruções do tipo *goto*. Não existe limites para identificadores. As variáveis devem começar com letras minúsculas ou sublinhado, e os tipos de parâmetros e retorno com letras maiúsculas. Não é necessário escrever explicitamente todos os tipos de um programa Haskell, mas se definindo torna-se uma documentação mais útil. Deve-se sempre seguir a tabulação durante a criação de suas funções, ou seja, devem estar na mesma coluna,

mas se necessário mais linhas para a definição da função, deve se utilizar uma coluna mais à direita, isso também conhecido como regras de layout. O if que é implementado através de |, significa a restrição do domínio do argumento da função, e é permitido o uso de mais de uma | (chamado de guards) durante as definições das funções, que se referem às suas condições. Mas há também a estrutura de if then e else. A extensão dos arquivos em Haskell é .hs.

Outro ponto é a equivalência entre matemática e Haskell, que pode ser vista em alguns exemplos a seguir:

Matemática:	$f(x)$	$f(x, y)$	$f(g(x))$	$f(x)g(y)$
Haskell:	<code>f x</code>	<code>f x y</code>	<code>f (g x)</code>	<code>f x * g y</code>

Haskell trabalha as matrizes como uma lista de listas, e são matrizes **retangulares**. É possível definir quaisquer limites da matriz, ou seja, não necessariamente os habituais índices (1,1) inicialmente, ficando a critério do programador. E é **homogênea**, já que todos os elementos da matriz devem ser do mesmo tipo e é possível identificar um elemento individualmente por sua posição, levando em consideração a posição do primeiro elemento da matriz. Para se criar uma matriz, utiliza-se um vetor de duas dimensões pela função array com uma tupla-2. Deve-se fazer o *import Data.Array* para se trabalhar com matrizes.

```
import Data.Array
```

```
nomeMatriz = array ((1,1), (2,2)) [((1,1),4), ((1,2),0), ((2,1),0), ((2,2),2)]
```

Onde ((1,1), (2,2)) definem os limites da matriz, no caso sua posição inicial e final.

Para se exibir os valores da matriz no terminal, basta utilizar o comando *elems nomeMatriz*.

```
elems nomeMatriz
```

```
Exibe: [4, 0, 0, 2]
```

Abaixo tem-se a Assinatura de funções: estrutura que deve ser utilizada para a criação de uma nova função em Haskell, lembrando que não é obrigatório escrever todos os tipos utilizados no programa, pois o compilador pode ajudar com esse empecilho.

```

1  --Assinatura de função:
2  somar :: Float -> Float -> Float
3  subtrair :: Float -> Float -> Float
4  multiplicar :: Float -> Float -> Float
5  dividir :: Float -> Float -> Float
6  radiciação :: Float -> Float -> Float
7  potenciação :: Float -> Float -> Float
8  par :: Int -> Bool
9  impar :: Int -> Bool
10 primo :: Int -> Bool
11 divisores :: Int -> [Int]

```

E na imagem a seguir, tem-se a forma que a função é criada, com seu nome e argumentos e em seguida, após a igualdade, a expressão do cálculo a realizar.:

```

12 --operações com dois números escolhidos:
13 somar a b = a + b
14 subtrair a b = a - b
15 multiplicar a b = a * b
16 dividir a b = a / b
17 potenciação a b = a**b
18 radiciação a b = potenciação b (dividir 1 a)
19 --operações com um número escolhido:
20 par a = a `mod` 2 == 0 -- "mod" é r
21 impar a = not (par a)
22 primo a = divisores a == [1,a] -- checar pr
23 divisores a = [b | b <- [1..a], a `mod` b == 0]

```

Outro ponto é a forma de comentar o código, que como mostram as imagens, é feito através dos símbolos "--", mas também pode ser feito entre "{- comentários -}".

4. FORMAS DE VINCULAÇÕES DE TIPOS

4.1 Clipper

Clipper possui uma tipagem dinâmica, porém surpreendente. No entanto, o clipper 5 introduziu a diretiva local, que permite declarar as variáveis no início da função ou procedimento. É um passo direto da tipagem estática.

Funções de manipulação de strings

1. ALLTRIM(<cTexto>)

- Descrição : Remove os espaços em branco em torno da string.
- Entrada : <cTexto> - Uma string qualquer.
- Saída : A mesma string sem os espaços em branco da direita e da esquerda.

2. LEN(<cTexto>)

- Descrição : Retorna o tamanho do texto especificado.
- Entrada : <cTexto> - Uma string qualquer.
- Saída : nTamanho - Um valor numérico inteiro.

3. UPPER (<cTexto>)

- Descrição : Retorna o texto fornecido totalmente em letras maiúsculas.260

Entrada : <cTexto> - Uma string qualquer.

- Saída : <cTextoMaiusculo>- A string escrita totalmente em letras maiúsculas.

4. LOWER(<cTexto>)

- Descrição : Retorna o texto fornecido totalmente em letras minúsculas.
- Entrada :<cTexto> - Uma string qualquer.
- Saída : <cTextoMinusc> -A string escrita totalmente em letras minúsculas.

5. LEFT(<cTexto>, <nQtd>)

- Descrição : Retorna caracteres a partir do primeiro caractere da esquerda.
- Entrada :
 - <cTexto> - Uma string qualquer.
 - <nQtd> <cTexto>- Quantidade de caracteres que devem ser retornados a partir da esquerda.
- Saída :
 - A string de entrada escrita somente com os caracteres iniciais.

6. RIGHT(<cTexto> , <nQtd>)

- Descrição : Retorna caracteres a partir do último caractere da esquerda.
- Entrada :
 - <cTexto> - Uma string qualquer.
 - <nQtd> - Quantidade de caracteres que devem ser retornados a partir da direita.
- Saída : - A string de entrada escrita somente com os caracteres finais.

7. SUBSTR(<cTexto> ,<nInicio> [,<nTamanho>])

- Descrição : Retorna os “n” caracteres a partir da posição <nInicio> . O tamanho da string de retorno é dada por [nTamanho]. Se [nTamanho] não for especificado, então retorna todos os caracteres de até o final da string.

- Entrada :

- <cTexto>- Uma string qualquer.

- <nInicio> - A posição inicial onde a substring inicia. nTamanho - O tamanho da substring.

- Saída :<cTextoRetorno> - A string de entrada escrita somente com os caracteres iniciais.

8. REPLICATE(<cTexto> ,<cTexto>)

- Descrição : Gera cópias sequenciais da string . A quantidade de cópias é definida por

- Entrada :

- <cTexto> - Uma string qualquer. – - Quantidade de cópias.

- Saída : <cTextoRetorno>- A string de entrada replicada vezes.

4.2 Java

O Java possui dois tipos de dados que são divididos **por valor** e **por referência**.

Os tipos primitivos são **boolean, byte, char, short, int, long, float e double**. Os tipos por referência, são classes que especificam os tipos de objeto **Strings, Arrays Primitivos e Objetos**. As variáveis primitivas quando não é especificado um valor são inicializadas por padrão. As variáveis de referência são inicializadas com o valor “**null**” (nulo).

A ligação estática em Java ocorre durante o tempo de compilação, enquanto a ligação dinâmica ocorre durante o tempo de execução.

private, finale static métodos e variáveis usam vinculação estática e são vinculados pelo compilador enquanto métodos virtuais são vinculados durante o tempo de execução com base no objeto de tempo de execução.

A ligação estática usa **Type**(**class** em Java) informações para ligação, enquanto a ligação dinâmica usa o objeto para resolver a ligação.

Métodos sobrecarregados são vinculados por meio de vinculação estática, enquanto métodos substituídos são vinculados por vinculação dinâmica em tempo de execução.

Java possui vinculações estáticas no tempo de implementação, no momento em que o programador especifica o tipo de dado e o modificador, se utilizado o modificador static, haverá uma vinculação de armazenamento antes da execução do programa. Como em outras linguagens também visto vinculação nos operadores aritméticos utilizados nas expressões, vinculação em tempo de execução na entrada de dados. Em Java, as variáveis definidas por padrão dinâmicas da pilha. Em Java todos objetos, exceto os tipos primitivos, são variáveis do tipo monte explícito.

As Unions consistem na união de valores de tipos distintos para formar um novo tipo de dados, que podem ser livres ou disjuntas. Java não suporta esse tipo.

4.3 Haskell

Sobre as vinculações de tipos de Haskell, a declaração do tipo acontece durante a criação da função, onde caso o programador especifique os “Tipos” que deverão ser utilizados pelos argumentos e o retorno esperado. Mas caso ele não os defina, a declaração de tipo será dada pelo próprio compilador, durante a argumentação para tal função, identificando os “Tipos” informados para saber com qual “Tipo” a função está trabalhando, em caso de discordância deles, ao compilar será emitido a notificação de erro, explicitando inclusive a variável diferente. No caso, é **tipo estático**, pois não será modificado em tempo de execução, isso acontece no momento de compilação.

Os tipos mais utilizados são:

Int, Integer, Float, Double, Char, String, Bool.

Em Haskell, alguns símbolos são também utilizadas na criação da função, como:

“::” significa “recebe”.

“->” significa “seguido de”, mas quando se tratar do último valor, significará “retorna”.

Alguns Operadores funcionam inclusive com uso de um símbolo, como:

+ Somar ***** Multiplicar **^** Exponenciação (int)

- Subtrair **/** Dividir ****** Exponenciação (float)

As unions não são definidas em Haskell. Já sobre caracteres, especialmente cadeias de caracteres, em Haskell os tipos utilizados são Char e String, para se utilizar char usa-se apenas ‘c’ (aspas simples), e para trabalhar com Strings usa-se aspas duplas “string”. É importante utilizar dessa maneira, pois em caso de utilizar, por exemplo: :type ‘aa’, será dado um erro de

sintaxe, já que o correto deveria utilizar: `:type "aa"`, para nesse exemplo descobrir o tipo de `aa`, que é uma `String`.

Há os seguintes tipos ordinais definidos pelo usuário em Haskell: enumeração, subintervalo. E funcionam da seguinte maneira:

Enumeração: É possível criar da seguinte maneira, e ao executar a `main`, será necessário apenas chamar a função `"num"` seguido do valor desejado, que esteja neste intervalo obviamente. Por exemplo ao solicitar no terminal:

num 2

Será então exibido `"Segunda"`, valor correspondente ao `num == 2`.

```
110 data Dia = Domingo
111     | Segunda
112     | Terca
113     | Quarta
114     | Quinta
115     | Sexta
116     | Sabado
117     deriving(Eq, Show)
118 dia :: Int -> Dia
119 dia num | (num == 1) = Domingo
120         | (num == 2) = Segunda
121         | (num == 3) = Terca
122         | (num == 4) = Quarta
123         | (num == 5) = Quinta
124         | (num == 6) = Sexta
125         | (num == 7) = Sabado
```

Subintervalo: Em haskell ao criar listas, é possível utilizar subsequências contíguas, como por exemplo pode-se criar uma da seguinte maneira:

listaAlfa = ['a'..'z']

E ao pedir para listar todos os elementos desta lista, serão exibidas todas as letras do alfabeto, de `"a"` a `"z"`. E também é possível utilizar algumas operações nestas listas, como por exemplo identificar o valor em uma `'n'` posição, por exemplo:

listaAlfa !!10

Será exibida a letra `"k"`, visto que começa de 0, logo será exibida a posição `10+1`.

5. CÓDIGO COMENTADO

5.1 Clipper

```
5  PROCEDURE Main
6  |
7  LOCAL nVar1, nVar2, nVar3, nCont := 1, nCont_res :=0 //valores a serem calculados
8
9  //recebendo dados do usuario
10 ? "Introduza dois numeros para que sejam feitas as operacoes"
11 INPUT "Introduza o primeiro valor : " TO nVar1
12 INPUT "Introduza o segundo valor : " TO nVar2
13
14
15 // Calculando e exibindo
16 ? " Soma ..... : " , nVar1 + nVar2
17 ? " Subtracao ..... : " , nVar1 - nVar2
18 ? " Multiplicacao .... : " , nVar1 * nVar2
19 ? " Divisao ..... : " , nVar1 / nVar2
20 ? " Potenciacao ..... : " , nVar1 ^ nVar2
21 ? " Radiciacao ..... : " , nVar1 ^ ( 1/nVar2)
```

Sobre o código

Parte 01:

Todas as linhas de comando são escritas dentro de uma função main, antecedida por PROCEDURE e encerrada com RETURN. Um ponto interessante é que a linguagem clipper não é case sensitive. Logo, os comandos acima poderiam ser escritos em caixa baixa. Na figura acima, têm-se as linhas de comando responsáveis pelos cálculos aritméticos em dois números. Existem inúmeras formas de declarar variáveis em clipper. Uma delas, é simplesmente declará-las e o valor atribuído a elas determina o seu tipo. Na linha 7, são declaradas as variáveis nVar1, nVar2, nVar3 e as variáveis contadoras nCont := 1, nCont_res :=0. Neste trecho de códigos, está sendo feito uso somente das variáveis nVar1 e nVar2. OS demais serão discutidas adiante. Têm-se um INPUT nas linhas 11 e 12, que estão concatenados a nVar1 e nVar2 respectivamente. Nas linhas de código consequentes, são realizadas as operações entre as variáveis que possuem os dados fornecidos pelo usuário. O comando "?" " mostra a mensagem na tela.

```

23 //Verificação de um numero(se ele é par ou impar e se é primo ou não)
24 ?
25 ? "Ensira um numero para verificar se ele eh par ou impar e se ele eh"
26 ? "primo ou nao."
27
28 INPUT "Ensira o numero que quer verificar : " TO nVar3
29 ?
30
31 /*Verificando se o numero é primo ou não*/
32
33 DO WHILE nCont <= nVar3
34
35     IF ((nVar3 % nCont) == 0)
36     |     ++ nCont_res
37     END IF
38
39     ++ nCont
40 END DO
41
42 if nCont_res == 2
43 | ? "O numero eh primo"
44 else
45 | ? "O numero nao eh primo"
46 end if

```

Parte 02:

A imagem acima mostra o trecho do código que verifica se o número inserido é primo ou não. Os dados fornecidos pelo usuário ficam armazenados na variável nVar3. Perceba que nCont e nCont_res são duas variáveis contadoras que são utilizadas dentro do laço. Ao sair do laço, faz-se uso do comando IF para verificação da contadora nCont_res. Dependendo do valor nela atribuído, pode-se verificar se o número é primo ou não.

```

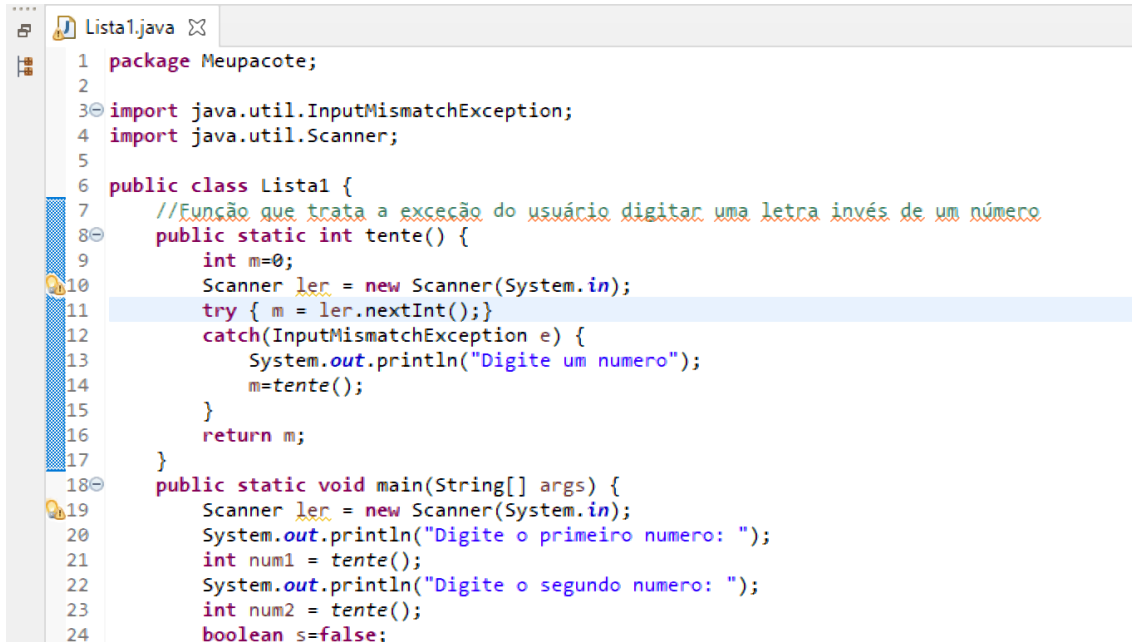
48 //verificando se o numero é par
49 IF ((nVar3 % 2) == 0) /* iniciação do if*/
50 | ? "O numero eh par."
51 ELSE
52 | ? "O numero eh impar."
53 END IF /*encerramento do if*/
54
55 INKEY(0) // pausar o programa
56
57 return

```

Parte 03:

O trecho de código acima é responsável por verificar se o número é par ou ímpar. Perceba que essa verificação é feita com os dados da variável nVar3, onde está fazendo uso do comando IF, onde, dependendo do valor do dado pertencente a nVar3, pode-se verificar se o número é par ou ímpar.

5.2 Java



```

1 package Meupacote;
2
3 import java.util.InputMismatchException;
4 import java.util.Scanner;
5
6 public class Lista1 {
7     //Função que trata a exceção do usuário digitar uma letra invés de um número
8     public static int tente() {
9         int m=0;
10        Scanner ler = new Scanner(System.in);
11        try { m = ler.nextInt();}
12        catch(InputMismatchException e) {
13            System.out.println("Digite um numero");
14            m=tente();
15        }
16        return m;
17    }
18    public static void main(String[] args) {
19        Scanner ler = new Scanner(System.in);
20        System.out.println("Digite o primeiro numero: ");
21        int num1 = tente();
22        System.out.println("Digite o segundo numero: ");
23        int num2 = tente();
24        boolean s=false;

```

Nesta parte inicial do código têm-se a chamada do pacote a qual o programa se encontra, a importação de superclasses uma de tratamento de entrada e outra para leitura de dados de entrada que são respectivamente `InputMismatchException` e `Scanner`, todas linhas de extrema importância para o código completo em si, pois alguns termos ganham significado exclusivo somente quando essas superclasses são importadas, caso contrário o compilador não entende o código corretamente e gera um erro.

Logo após as definições do pacote e a importação das superclasses têm-se a definição das classes que estão sendo trabalhadas no momento, possui um modificador que permite que todas classes do mesmo pacote a acessem e o termo reservado que lhe define como classe, além de sua nomeação que é `Lista1`. Dentro do escopo da `Lista1` há duas funções, uma voltada para tratamento que poderia ser escrita dentro da outra, a qual é a `main`, mas por organização no código para uma reutilização de código, pois pode ser chamada em qualquer parte da `main`.

A função ‘tente’ serve para tratar uma exceção gerada na inserção de um tipo errado de dado, quando o programa solicita um inteiro e o usuário digita uma letra, a mesma lê um número e caso aconteça a exceção ela faz um efeito recursivo até que o usuário digite o que é solicitado.

Ainda na mesma captura de tela é possível ver o início da função `main`, na linha de definição da `main` é possível que ela pode ser acessada livremente de qualquer pacote por conta do modificador `public`, que não precisa ser instanciada ao ser chamada, `static` e que não retorna um valor.

```

18 public static void main(String[] args) {
19     System.out.println("Digite o primeiro numero: ");
20     int num1 = tente();
21     System.out.println("Digite o segundo numero: ");
22     int num2 = tente();
23     boolean s=false;
24     System.out.println("\nDigite a operacao desejada:\n Soma (1);\n Subtracao (2);\n
25     int oper = tente();
26     do {
27         switch (oper) {
28             case 1:
29                 System.out.println(num1 + " + " + num2 + " = " + (num1 + num2) );
30                 s=true;
31                 break;
32             case 2:
33                 System.out.println(num1 + " - " + num2 + " = " + (num1 - num2) );
34                 s=true;
35                 break;
36             case 3:
37                 System.out.println(num1 + " * " + num2 + " = " + (num1 * num2) );
38                 s=true;
39                 break;
40             case 4:
41                 System.out.println(num1 + " / " + num2 + " = " + (num1 / num2) );
42                 s=true;
43         }

```

Após declarar a main, são solicitados os dois números, as variáveis recebem os valores por meio de chamadas da função tente que é voltada para o tratamento entrada de letra no lugar de um inteiro, a seguir tem a variável booleana “s” que serve para verificar quando prosseguir com a execução do código além do switch ou não, o switch identifica qual operação deve ser realizada com os dois números por meio da variável inteira, mostra a notação infixada e indica que o código deve prosseguir por denominando “s” como true, caso o número digitado seja uma das operações listadas, caso não seja, a estrutura de repetição vai iterar com base no valor da variável s pedindo que o usuário digite um valor listado.

```

        s=true;
        break;
    case 3:
        System.out.println(num1 + " * " + num2 + " = " + (num1 * num2) );
        s=true;
        break;
    case 4:
        System.out.println(num1 + " / " + num2 + " = " + (num1 / num2) );
        s=true;
        break;
    case 5:
        System.out.println("Raiz de " + num1 + " de " + num2 + " = " + (Math.pow(num1, (1.
        s=true;
        break;
    case 6:
        System.out.println(num1 + " elevado a " + num2 + " = " + (Math.pow(num1, num2);
        s=true;
        break;
    default:
        System.out.println("Digite um dos numeros listados!");
        System.out.println("\nDigite a operacao desejada:\n Soma (1);\n Subtracao (2);\n Mu
        oper = tente();
        s=false;
    }
}while(s==false);

```

E por fim a lógica de verificar se um número é par ou ímpar e se um número é primo e o fim dos escopos da main e da classe Lista1 representado pelas chaves.

```
61      System.out.println("Digite um numero");
62      int x = tente();
63      if(x % 2 == 0)
64          System.out.println("O numero e par");
65      else
66          System.out.println("O numero e impar");
67
68      System.out.println("Digite um numero:");
69      int p = tente();
70      int d = 2;
71      int cd = 0;
72
73      //Verifica se um número é primo pelo resto da divisão inteira e pela quantidade d
74      while(d<=(int)(Math.sqrt(p)) && cd==0 ){
75          if(p % d == 0)
76              cd++;
77          d++;
78      }
79      if(cd>1)
80          System.out.println("Nao e primo");
81      else
82          System.out.println("E primo");
83  }
84
85  }
86  }
```

5.3 Haskell

Com o programa “Teste.hs” já em execução, basta executar a main e seguir as instruções. Para realizar tais tarefas, foi necessário instalar o GHCi na máquina. O GHCi que é o compilador e ambiente interativo livre para Haskell. GHC: Glasgow Haskell Compiler.

Código na íntegra abaixo (ênfase para o importantíssimo espaçamento aplicado).

```
--Assinatura de função:
somar :: Float -> Float -> Float           --nomeFunção
:: TipoRecebe -> TipoRecebe -> TipoRetorno
subtrair :: Float -> Float -> Float
multiplicar :: Float -> Float -> Float
dividir :: Float -> Float -> Float
radiciação :: Float -> Float -> Float      --Função ::
indiceRaiz -> valorRaiz -> Retorno
potenciação :: Float -> Float -> Float
par :: Int -> Bool                        --par e impar
já existem nativamente como: even e odd, respectivamente
impar :: Int -> Bool
primo :: Int -> Bool
```



```

divisores :: Int -> [Int]
--operações com dois números escolhidos:
somar a b = a + b --estrutura da
função soma: nomeFunção argumento1 argumento2
subtrair a b = a - b
multiplicar a b = a * b
dividir a b = a / b
potenciação a b = a**b -- usa-se "a**b"
para se trabalhar com Float; "a^b" para Int
radiciação a b = potenciação b (dividir 1 a) -- a: indice,
b: radicando; 1ro: dividir 1 a, 2do: potenciação b**1/a
--operações com um número escolhido:
par a = a `mod` 2 == 0 -- "mod" é nativa e serve
pra calcular o resto da divisão
impar a = not (par a)
primo a = divisores a == [1,a] -- checar primo: como não
há "loops", deve-se verificar com outra função, a divisores, se
existe outro div além de [1,a]
divisores a = [b | b <- [1..a], a `mod` b == 0] -- essa função
"lista" todos os divisores de 'a', criada para utilizar a função
primo

main = do
putStrLn "Deseja fazer operações com '1' ou '2' números?" --
definir se será 1 ou 2 números a serem utilizados
n <- readLn
if n == 1
then do putStrLn "Digite um número inteiro: (definir:
par/impar/primo)" -- quando n == 1
num <- readLn
if (par num) -- função par com num como
argumento
then putStrLn "é par"
else putStrLn "é ímpar"
if (primo num) -- função primo
then putStrLn "e é primo"
else putStrLn "e não é primo"
else if n == 2 --
quando n == 2
then do putStrLn "(operações com dois valores)"
print "Digite o primeiro valor: "
valor1 <- getLine
print "Digite o segundo valor: "

```

```

        valor2 <- getLine
-- utilizando as funções já criadas:
        putStrLn ("Soma: " ++ valor1++"+"++valor2++" =
"++show(somar (read valor1) (read valor2)))
-- função somar
        putStrLn ("Subtracao: "++valor1++"-
"++valor2++" = "++show(subtrair (read valor1) (read valor2)))
-- função subtrair
        putStrLn ("Multiplicacao:
"++valor1++"*"++valor2++" = "++show(multiplicar (read valor1) (read
valor2)))
-- função multiplicar
        putStrLn ("Divisao: "++valor1++"/"++valor2++"
= "++show(dividir (read valor1) (read valor2)))
-- função dividir
        putStrLn ("Radiciacao: (radicando:
"++valor2++" e índice: "++valor1++) raiz = "++show(radiciação (read
valor1) (read valor2))) -- função radiciação
        putStrLn ("Potenciacao:
"++valor1++"^"++valor2++" = "++show(potenciação (read valor1) (read
valor2)))
-- função potenciação
        else putStrLn "Digite apenas '1' ou '2'" -- n é diferente de
1 ou 2 ;/

```

6. REFERÊNCIA BIBLIOGRÁFICA

- [1] WIKIPÉDIA, **Haskell (linguagem de programação)**. Disponível em: [https://pt.wikipedia.org/wiki/Haskell_\(linguagem_de_programação\)](https://pt.wikipedia.org/wiki/Haskell_(linguagem_de_programação)) Acesso em: 05 de julho de 2021.
- [2] WIKIPÉDIA, **Clipper (linguagem de programação)**. Disponível em: [https://pt.wikipedia.org/wiki/Clipper_\(linguagem_de_programação\)](https://pt.wikipedia.org/wiki/Clipper_(linguagem_de_programação)) . Acesso em: 06 de julho de 2021.
- [3] PACIEVITCH, Yuri. **História do Java**. Disponível em: <https://www.infoescola.com/informatica/historia-do-java/> . Acesso em: 06 de julho de 2021.
- [4] NOLETO, Cairo. **Paradigmas de programação: o que são e quais os principais?**. Disponível em: <https://blog.betrybe.com/tecnologia/paradigmas-de-programacao/#:~:text=Paradigma%20orientado%20a%20objetos,multiplataforma%20d e%20uma%20mesma%20maneira.&text=Alguns%20exemplos%20de%20linguagens%20orientadas,C%2B%2B%2C%20C%23%20e%20Python.> . Acesso em: 06 de julho de 2021.
- [5] MARLOW, Simon. **Fighting spam with Haskell**. Disponível em: <https://engineering.fb.com/2015/06/26/security/fighting-spam-with-haskell/> . Acesso em 09 de julho de 2021.
- [6] MARLOW, Simon. **Haskell in the Datacentre**. Disponível em: <https://simonmar.github.io/posts/2016-12-08-Haskell-in-the-datacentre.html> Acesso em 09 de julho de 2021.
- [7] SWIFT NAVIGATION. **Swift Binary Protocol**. Disponível em: <https://support.swiftnav.com/support/solutions/articles/44001850782-swift-binary-protocol> Acesso em 09 de julho de 2021.
- [8] L. Erkök et al.:**Programming Cryptol**. Galois, 2015.
- [9] CASTRO, Marcos. **Curso de Haskell - Aula 55 - Enumeração (enum)**. Youtube, 20 de novembro de 2014. Disponível em: <https://www.youtube.com/watch?v=FQFpoNEZ1lo> . Acesso em 09 de julho de 2021.
- [10] Fundamentos da Linguagem Java, **IBM**, 2020. Disponível em: [Fundamentos da linguagem Java – IBM Developer](#) Acesso em 11 de julho de 2021.