Cartão de Referência

Java Básico (J2SE 1.4)

versão 1.02

Fundamentos

- Todo arquivo possui a extensão .java;
- Contém uma classe definida como pública (com nome idêntico ao arquivo) e ilimitadas classes particulares;
- A classe se apresenta na seguinte ordem: Declaração package, Declarações import e Declarações da Classe; e

```
Para o arquivo: Test.java
package exame.guia;
import java.util.*;
public class Teste { ... }
class Teste2 { ... }
```

- O método main é o ponto de entrada da aplicação:

```
public static void main(String[] args)
```

- Ele é declarado público por convenção e static por necessidade;
- Recebe parâmetros através da chamada a classe:

```
java Paises Brasil EUA
```

- args[0] terá "Brasil" e args[1] terá "EUA".

Palavras Chaves e Reservadas

Nomes iniciam por letras (a..z A..Z), sinal de dolar ($) ou underscore (_) depois a estes é permitido dígitos (0..9), também não é permitido o uso das seguintes palavras:

```
abstract     assert      boolean     break       byte
case         catch       char        class       continue
default      do          double      else        extends
false        float       final       finally     for
if           implements  import      instanceof  int
interface    long        native      new         null
package      private     protected   public      return
short        static      strictfp    super       switch
synchronized this        throw       throws      transient
true         try         void        volatile    while

*const            *goto
```

Tipos Primitivos

boolean (1 bit) true ou false – boolean b = true;
byte (8 bits) -2^7 a 2^7-1 – byte b = 0;
short (16 bits) -2^{15} a $2^{15}-1$ – short s = 0;
char (16 bits) 0 a $2^{16}-1$ – char c = 0; 'a' ou '\u0000'
int (32 bits) -2^{31} a $2^{31}-1$ – int i = 0;
long (64 bits) -2^{63} a $2^{63}-1$ – long l = 0L;
float (32 bits) -10^{38} a 10^{38} – float f = 0.0F;
double (64 bits) -10^{308} a 10^{308} – double d = 0.0;

- Float (7 dig) e double (15 dig) são tipo IEEE 754 variando de: NaN – Not a Number, NEGATIVE_INFINITY a POSITIVE_INFINITE
- O sufixo F ou f (1.82f) indica float literal, D ou d indica double literal (1.82d) e um L ou l indica um long (12L).

Operadores e Assinaladores

- **Unários**

```
Incrementos e Decrementos (a esquerda participa):
Inicial x       Expressão       Final Y         Final X
5               y = x++;        5               6
5               y = ++x;        6               6

Operador de inversão ( ~ )
i = ~10  ==> -11  ==> fórmula: (-x)-1

Operador de negação ( ! )
if (!true)  ==> false

Operador de coerção de tipos
float a = (float)1.2;
```

- **Aritméticos**

```
Multiplicação e Divisão
int a = 12345, b = 234567;
int c = a * b / b;  ==>  resp. -5965

Operador Módulo ( % )
17 % 5 => 17 - 5 => 12 - 5 => 2
7.6 % 2.9 => 7.6 - 2.9 => 4.7 - 2.9 => 1.8

Adição e Subtração
int a = 1, b = 2;
String c = "" + a + b  => 12
```

- **Deslocamento**

x<<y - Desloca os bits de 'x' para a esquerda pelo número de bits especificado por 'y', aumentando o valor do número e preenche a partir da direita com 0s (52<<2 = 208);
x>>y - Desloca os bits de 'x' para a direita pelo número de bits especificado por 'y', Se 'x' for negativo preenche com 1s a partir da esquerda, caso contrário com 0s, diminuindo o valor do número (52>>3 = 6); e
x>>>y - Desloca os bits de 'x' para a direita pelo número de bits especificado por 'y', 0s são inseridos a partir da esquerda.

- **Comparação**

Operadores matemáticos: < (menor), <= (menor igual), > (maior) e >= (maior igual)

instanceof – testa se o objeto pertence a classe em questão.

== (igual) != (diferente) utilize para atributos

Utilize o método equals(String) para objetos.

- **Comparação Binária**

```
& - uma coisa e outra
0 & 0 = 0; 0 & 1 = 0; 1 & 0 = 0; 1 & 1 = 1

^ - uma coisa ou exclusivamente a outra
0 ^ 0 = 0; 0 ^ 1 = 1; 1 ^ 0 = 1; 1 ^ 1 = 0

| - uma coisa ou a outra
0 | 0 = 0; 0 | 1 = 1; 1 | 0 = 1; 1 | 1 = 1
```

- **Comparação Lógica (Curto Circuito)**

```
&& - uma coisa e outra
F && F = F; F && T = F; T && F = F; T && T = T

|| - uma coisa ou a outra
F || F = F; F || T = T; T || F = T; T || T = T
```

- **Condicional**

```
(condicao)?true:false

Por exemplo:
int x = 4;
((x > 4)?10.0:5); ==> Resposta 5.0
```

- **Assinaladores**

```
byte x = 2; x += 3;  ==> 5
byte x = 2; x = (byte)(x + 3)  ==> 5
int a, b, c;  a = b = c = 0;

*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=
```

Classe Array

- Ao criar um array: Declare, Construa e Inicialize
- Cada conjunto de colchetes define uma dimensão do array;
- Valores iniciais dos tipos primitivos:

```
boolean (false)          char ('\u0000')
byte (0)                 short (0)
int (0)                  long (0L)
float (0.0f)             double (0.0d)
```

- Declarado de objeto inicializa automaticamente com "null"
- Índice do array é iniciado com [0] e o atributo length possui seu tamanho.

Modificadores de Acesso

- **public**

livre acesso por qualquer outra classe, obrigatório para as applets.

- **protected**

livre acesso para classes do mesmo pacote.

- **private**

acesso apenas para a classe que o definiu, não é propagado por herança

- Override em métodos

private -> [Default] -> protected -> public

Outros Modificadores

final – classes, métodos e variáveis, não poderão mais serem modificados;
abstract – classes e métodos, não são instanciados, serão utilizados por herança, declarado sem corpo;
static – métodos e variáveis, não ocorre a subscrição dos métodos (utilizada em Serializable);
native – corpo do método encontrado fora da jvm;
transient – variáveis não são armazenadas como parte do objeto;
synchronized – usado para controlar o acesso multi-threaded dos programas; e
volatile – usado para variáveis funcionando de forma contrária ao synchronized.

Fluxo de Controle

- **Condicional**

```
Condicional "if"
if (condição) { ... }  else { (...) }

Condicional "switch"
switch (inteiro) {
  case val1: ...; break
  case val2 + 1: ...; break
  default: ...;
}
```

- **Repetição**

```
Laço "while"
while (condição) { ... }

Laço "do ... while"
do { ... } while (condição);

Laço "for"
for (declaração; condição; expressão) { ... }
```

- **break**

interrompe a ação do laço.

- **continue**

abandona a execução e retorna ao cabeçalho do laço, ou a um label determinado.

```
lacoPrinc: for (int i = 0; i < 3; i++)
  for (int j = 0; j < 2; j++)
   if (i == j) continue lacoPrinc;
```

- **Proteção**

Usando "try" e "catch"

```
try {
  comandos protegidos
} catch ([classe exceção] [var exceção]) {
  caso em erro
}
```

Usando "finally" - acontecendo ou não o erro.

```
try {
  comandos protegidos
} catch ([classe exceção] [var exceção]) {
} finally {
}
```

Threads

- Como executar uma thread

Utiliza-se o método start(), e a partir deste é disparado o método "public void run()"

- Finalizar a execução da thread

Uma thread finaliza a execução quando todos os comandos do método run() são executados.

- Prioridade

Modificada pelo método setPriority(int) e sendo obtida pelo método getPriority(int), variando entre Thread.MIN_PRIORITY, Thread.NORM_PRIORITY e Thread.MAX_PRIORITY.

- Estados da thread

Assume os seguintes estados: Running, Waiting, Sleeping, Suspended, Blocked, Ready e Dead.Estado Ready - método yield faz com que o estado da Thread torne-se reading

- Estado Waiting - método wait e notify

interrompe a ação da thread por um tempo entre a chamada método wait() e sua finalização com o método notify() ou notifyAll()

- Estado Sleeping - método sleep(long milisegundos)

interrompe a ação da thread por um tempo definido

- Estado Blocked - acontece com Socket

É iniciado durante um processo de E/S

- Conceito de monitor

Um monitor é um objeto que pode bloquear e reviver uma thread, e deverá ser chamado pela própria thread.

- Criando um monitor

Cria-se uma classe com métodos sincronizados que fazem chamadas aos métodos wait() e notify():

```
class Monitor
  public synchronized void parar() {
    try {
      wait();
    } catch (InterruptedException e) { }
  }
  public synchronized void iniciar() {
    notify();
  }
}
```

Manipulando objetos da java.util.Date

- Objetos no formato Data em Java são gerados pela classe *Date* e a classe *SimpleDateFormat* determina os formatos de saída.

Formatos:

```
G - Era (AD)
y - Ano (2001)
M - Mês (july 07)
d - Dia (10)
h - Hora am/pm (1 a 12)
H - Hora (0 a 23)
m - Minutos (30)
s - Segundos (1)
S - Milisegundos (1)
E - Dia da semana (Tuesday)
D - Dia do ano (189)
F - Dia da semana dentro do mês (2 - 2nd Wed in July)
w - Dia da semana no ano (27)
W - Dia da semana no mês (2)
a - Am/Pm (AM)
k - Hora (1 a 24)
K - Hora am/pm (0 a 11)
z - Zona (Pacific Standard Time)
' - Para digitar o texto ('às')
```

Exemplo:

```
import java.util.*;
import java.text.*;
public class TestDtHr {
  public static void main(String args[]) {
    long aHora = System.currentTimeMillis();
    Date aData = new Date();
    System.out.println("Hora: " + aHora);
    System.out.println("Data: " + aData);
    System.out.println("Data: " +
    (new SimpleDateFormat("'Dia' dd/MM/yyy")).format(aData));
    System.out.println("Hora: " +
    (new SimpleDateFormat("'às' hh:mm")).format(aData));
  }
}
```

Formatação de Números

Formatos:

```
# - Substituição dos números
0 - Substituição dos números colocando "0" quando vazio
$ - Sinal de moeda
, - Separador de grupo (trocado pelo método setGroupingSeparator)
. - Separador Decimal (trocado pelo método setDecimalSeparator)
```

Exemplo:

```
import java.text.*;
public class TestNumber {
  public static String formValor(String masc, double num) {
    DecimalFormatSymbols dfs = new DecimalFormatSymbols();
    dfs.setDecimalSeparator('/');
    dfs.setGroupingSeparator('-');
    DecimalFormat df = new DecimalFormat(masc, dfs);
    return df.format(num);
  }
  public static void main(String args[]) {
    System.out.println("Valor 1: " +
      formValor("####.##", 123456.789));
    System.out.println("Valor 2: " +
      formValor("###,###.##", 123456.789));
    System.out.println("Valor 3: " +
      formValor("$###,###.##", 123456.789));
    System.out.println("Valor 4: " +
      formValor("00000000.000", 123456.789));
  }
}
```

Objetos da java.lang

- **Classe Math**

```
abs(x)   - valor absoluto de x : depende entrada
ceil(x)  - primeiro inteiro maior que x : double
floor(x) - primeiro inteiro menor que x : double
max(x,y) - maior valor entre x e y : depende entrada
min(x,y) - menor valor entre x e y : depende entrada
random() - núm. Aleatório >= 0.0 < 1.0 : double
round(x) - arredondamento (.5 para mais) : long
sin(x)   - seno de x : double
cos(x)   - cosseno de x : double
tan(x)   - tangente de x : double
sqrt(x)  - raiz quadrada de x : double
pow(x,y) - x elevado a potência y : double
exp(x)   - exponencial : double
log(x)   - logaritmo natural de x (base e) : double

Constantes
PI       - valor 3.141592653589793
E        - valor 2.718281828459045
```

- **Classe String**

```
charAt(pos)     - devolve o char da posição
concat(Str)     - concatena Str com o objeto
compareTo(Str)  - 0 se = , <0 se Str< ou >0 se Str>
endsWith(Str)   - retorna true com Str no final
equals(Str)     - compara se duas str são idênticas
equalsIgnoreCase - compara se duas str são iguais
indexOf(ch)       - retorna o indice do char
lastIndexOf(ch)   - retorna o indice do char
length()          - tamanho do objeto
replace(ch1,ch2)  - troca de letras
startsWith(Str)   - retorna true com Str no início
substring(in,fi)  - retorna a subcadeia
toLowerCase()     - letras em minúscula
toUpperCase()     - letras em maiúscula
trim()            - retira os espaços em branco
```

Observações