Capítulo 3 - Fundamentos da Linguagem Java

-> Principais características do linguagem Java:

-> Diferencia caracteres maiúsculos de minúsculos (Case-sensitive)
-> Blocos de código são definidos entre chaves {}
-> Ao final de cada instrução é obrigatório o uso do ponto e vírgula
-> A classe deve ser salva em um arquivo com o mesmo nome da classe e extensão .java
-> Todo programa em Java é representado por uma ou mais classes
-> Normalmente trabalha-se com uma única classe por arquivo

-> Declaração de classes, variáveis, atributos e métodos:

-> Ao declararmos classes, métodos ou variáveis, devemos seguir algumas regras:
    -> O identificadores devem começar com letras A-Z, a-z, _ ou $
    -> A partir do segundo caracter, podem ser usados números
    -> Não podem ser usadas palavras reservadas

-> Declaração de classes:
-> A declaração de classes é feita utilizando-se a palavra reservada 'class' seguida de um identificador válido e do par de chaves.
-> Para que uma classe possa ser executada pela JVM, é necessário declarar um método especial comumente conhecido como método main:

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

-> Um pouco de código (Exemplos)

-> Palavras reservadas: (p. 40)

-> Não podem ser usadas como identificador. Exemplos

-> Convenções de código:

-> A Sun sugere algumas convenções que devem ser seguidas para que o código tenha maior legibilidade e maior compreensão.

-> Nome de classe: Devem começar com uma letra maiúscula e se for um identificador formado por mais de uma palavra, cada nova palavra deve ter a primeira letra em maiúsculo. Exemplos

-> Métodos e variáveis: A primeira palavra deve ser minúscula, e cada nova palavra ter a primeira letra maiúscula. Exemplos

-> Essa convenção é conhecida como CamelCase

-> Constantes: Devem apresentar todos os caracteres maiúsculos e a separação de palavras utilizando underscore. Exemplo

-> Identação: Tabulações com 4 espaços.

-> Comentários:

-> Comentários inline: Comentários de uma única linha são representados por "//".

-> Comentários multiline: Comentários de múltiplas linhas devem ser iniciados com /* e finalizados com */

-> Comentários Javadoc: São parecidos com comentários multiline, porém são representados por

/** */

-> A partir de comentários Javadoc pode ser gerada uma documentação em formato HTML

-> Algumas tags padrão podem ser usadas para auxiliar na documentação: @author, @revision, @return

-> A documentação das APIs é conhecida como javadoc e é gerada através do utilitário javadoc:
    javadoc -private -author TesteJavadoc.java

Capítulo 4 - Variáveis

-> Variáveis representam espaços de memória para armazenar valores.

-> Para cada variável, devemos definir um tipo e um nome (um identificador válido)

-> Em Java as variáveis podem ser de dois tipos: primitivos e referência

-> Tipos primitivos:

-> Sintaxe para declaração e inicialização

-> Tipos primitivos podem ser Numéricos, Caracteres ou Booleanos

-> Números primitivos: byte, short, int e long (p. 52)

-> O primeiro bit da esquerda é utilizado para o sinal + ou -

-> Os literais numéricos podem ser expressos no sistema decimal, octal(0 a 7), ou hexadecimal (0-9 e a-z ou A-Z)

-> Para indicar um número em octal, devemos prefixá-lo com 0

-> Números hexadecimais são prefixados com 0x ou 0X.

-> Todo número inteiro em Java é tratado como int, desde que o valor esteja na faixa de intervalos do int

-> Para forçar um número para inteiro para long podemos usar o sufixo l ou L

-> Para forçar um número para byte ou short, devemos fazer um cast

-> Exemplos (Utilizando valores fora da faixa)

-> Números com ponto flutuante:

-> Existem dois tipos de variáveis para números com pontos flutuantes: float e double (p. 54)

-> Todo número com ponto flutuante por padrão é double

-> Podemos utilizar o sufixo d ou D para indicar explicitamente que um número é double

-> Para indicar que um número é do tipo float, deve-se utilizar o sufixo f ou F após o número.

-> Como todo número flutuante é double por padrão, e o repositório de um float é menor que um double, não podemos atribuir um literal de ponto flutuante diretamente a uma variável float

-> Exemplos

-> Caracteres:

-> Os caracteres são representados pelo tipo primitivo char.

-> As variáveis do tipo char ocupam 16 bits de memória

-> Literais:

-> Literais char podem ser expressos incluindo o caracter desejado entre aspas simples

-> Outra maneira de expressar um literal caracter é especificar o código Unicode: quatro dígitos em hexadecimal precedidos por \u

-> Java suporta também uma série de sequencia de escape usando a barra invertida:
    \n -> Nova linha
    \' -> Aspas simples
    \" -> Aspas duplas
    \t -> Tabulação
    \\ -> Barra invertida

-> Exemplos

-> Booleanos:

-> Variáveis de tipos booleanos podem ser representadas apenas por dois valores: true ou false

-> Variáveis booleanas ocupam apenas um bit

-> Não existe nenhum mapeamento entre true e false e os números inteiros como em outras linguagens

-> Exemplos

-> Tipos reference:

-> Variáveis do tipo reference armazenam o endereço de memória para um determinado objeto

-> A quantidade de memória varia de acordo com o tamanho do objeto

-> Sintaxe para declaração e inicialização de uma variável de tipo reference

-> Variáveis do tipo reference podem conter os seguintes valores:
    - null
    - referência para um objeto compatível com o tipo declarado
    - referência para um array

-> Literais:
 -> Somente o valor null, indicando que a variável não referencia nenhum objeto ou array

-> Exemplos

-> A classe String:
  -> A classe String é uma das primeiras a ser utilizada devido ao fato de poser ser inicializada de maneira semelhante aos tipos primitivos

-> Variáveis locais:
  -> Variáveis declaradas dentro de métodos ou blocos de código são definidas como locais

  -> Esse tipo de variável não possui valor de inicialização padrão.

  -> Se declararmos uma variável local sem informar um valor, e tentarmos usá-la, teremos um erro de compilação

  -> Se apenas declararmos a variável sem in formar um valor e não utilizarmos esta variável, o compilador não irá se importar

  -> Exemplos

-> Escopo:
  -> O escopo define em qual parte do programa a variável estará disponível

  -> O escopo da variável geralmente é definido pelo par de chaves no qual a variável se encontra

  -> Se tentarmos acessar uma variável fora do escopo, teremos um erro de compilação

Capítulo 5 - Operadores

-> São símbolos previamente definidos na linguagem de programação para representar operações aritméticas, operações de conversão de tipos etc.

-> Operadores unários:
  -> Operadores unários são aqueles que envolvem apenas um operando:
  - Negação: !
  - Pré e pós incremento: ++
  - Pré e pós decremento: --
  - Sinal positivo: +
  - Sinal negativo: -
  - Cast ()

  -> Operador de negação: !
    -> É utilizado para inverter o valor de uma expressão booleana. !false resulta em true e !true resulta em false

    -> Exemplo

  -> Operador de incremento e decremento: ++ e --
    -> Estes operadores modificam o valor de uma variável adicionando ou subtraindo 1

    -> Operadores de incremento e decremento podem ser:
      - Pós-fixados: Primeiro a variável é usada e depois seu valor é alterado
      - Pré-fixados: O valor é alterado antes de a variável ser usada

    ->Exemplos

  -> Operadores de representação de sinal: + e -
    -> Representam o positivo e o negativo respectivamente
    -> O operador + apenas deixa explícito que o número é positivo
    -> Exemplos

-> Operador de conversão: Cast
  -> O operador de cast é usado para converter tipos
  -> Pode ser utilizado para troca de tipos entre valores primitivos
  -> Também pode ser utilizado para o cast de objetos
  -> O operador de conversão pode ser explícito ou automático
  -> O cast explícito deve ser usado quando queremos atribuir uma variável 'maior' a uma variável 'menor'
  -> Quando queremos atribuir uma variável 'menor' a uma 'maior', o cast acontece automaticamente
  -> A operação de cast deve ser usada com cuidado, pois, ao convertermos tipos primitivos podemos perder a precisão dos valores
  -> Exemplos

-> Operadores aritméticos: +, -, *, /, e %
  -> Envolvem dois ou mais operandos e representam operações matemáticas

  -> Adição e subtração: + e -
    -> O operador + é sobrecarregado na linguagem Java

  -> Multiplicação e Divisão: * e /

  -> Resto da Divisão: %
      -> Também conhecido como operador de módulo. Retorna o resto da divisão

-> Operadores de Comparação: <, <=, >, >=, == e !=
  -> Os operadores de comparação são otilizados para comparar valores e retornam true ou false

  -> Exemplos

  -> A comparação de igualdade, entre os tipos primitivos, deve ser sempre efetuada com o operador de comparação

-> Operadores de comparação de tipos: instanceof
  -> O operador instance of verifica se um determinado objeto é uma instância de uma classe

-> Operadores Lógicos:
  -> Operadores AND e OR (&& e ||)
   -> Esses operadores são aplicáveis somente entre operandos booleanos

  -> Operador AND (&&)
   -> O resultado de uma operação usando este operador só será true se as duas condições forem verdadeiras

   -> A segunda condição só será analisada se a primeira condição retornar true

  -> Operador OR (||)
    -> Se um dois operandos for true, o resultado será true

    -> Se o resultado da primeira expressão retornar true, a segunda expressão não será executada;

  -> Operadores bit a bit: &, | e ^
    -> Os operadores & e | seguem as mesmas regras de && e ||, porém sempre irão verificar as duas condições

-> O operador ^ irá retornar true se as condições forem true e false ou false e true, caso contrário irá retornar false

-> Operadores de Atribuição: =, +=, -+, *=, /= e %=
-> Estes operadores atribuem um novo valor a uma variável ou expressão:

-> O operador = é usado para uma atribuição simples: x = 7;
-> O perador +=, incrementa o valor da variável com o operando ou expressão do lado direito
-> Os operadores -=, *= e /= são análogos ao operador +=

-> Operador Ternário
-> O operador ternário atribui novos valores a uma variável com base em um comando condicional
-> Exemplos

-> Capítulo 6 - Controle de fluxo

-> if, else
-> Os comandos condicionais if/else são usados para controlar o fluxo que o programa irá seguir
-> Sintaxe
-> O uso de chaves é opcional quando temos apenas uma instrução a ser executada pelo comando condicional
-> A instrução avaliada dentro do if deve obrigatoriamente ser ou retornar um valor booleano:
-> Valores literais true ou false;
-> Uma variável booleana
-> Uma expressão que retorne true ou false
-> A chamada a um método que retorne um booleano
-> A expressão a ser avaliada no if, deve estar SEMPRE entre parênteses
-> A instrução else nunca deve ser seguida por uma nova avaliação
-> É possível utilizar expressões if sem um correspondente else
-> Não é possível usar uma expressão else sem um if correspondente

-> switch
-> É um comando de controle de fluxo similar ao if/else
-> Quando um quantidade muito grande de if/else precisar ser usada, podemos substituir por uma instrução switch
-> Sintaxe. (Exemplo)
-> Apenas os seguintes tipos podem ser avaliados pelo comando switch:
- char
- byte
- short
- int
- enum (módulo2 - capítulo 9)
-> A cláusula case suporta apenas valores literais ou constantes de tipos compatíveis com a variável declarada na cláusula switch
-> Quando um case verdadeiro é encontrado, o programa segue executando as instruções do switch até encontrar uma instrução break ou o término do bloco switch
-> A cláusula default não é obrigatória e será executada quando não for encontrado um case específico
-> A clásula switch não permite testar intervalos de valores
-> Acláusula default não precisar ser a última instrução do bloco switch
-> Exemplo. (Imprimir o dia da semana por extenso)

-> while
  -> O comando while é um comando de repetição que é utilizado para determinar de forma lógica quantas vezes desejamos executar um bloco de instruções
  -> Sintaxe
  -> O uso de chaves é opcional quando temos apenas uma instrução a ser executada pelo comando de repetição

-> do while
  -> Parecido com o while, porém o bloco de instrução será sempre executado uma vez antes do teste while, mesmo que o teste while seja false
  -> Sintaxe
  -> O uso de chaves é opcional quando temos apenas uma instrução a ser executada pelo comando de repetição
  -> IMPORTANTE: Não esquecer do ; depois do comando while

-> for
  -> No comando de repetição for, uma determinada expressão será repetida até que a condição do for não seja mais verdadeira.
  -> Diferentemente do while, o for é geralmente usado quando queremos percorrer um conjunto de valores previamente conhecidos.
  -> Sintaxe
  -> O uso de chaves é opcional quando temos apenas uma instrução a ser executada pelo comando de repetição
  -> A estrutura do comando for, é dividida em três partes:
    -> Inicialização:
      -> É a expressão que inicializa o loop e será executada uma única vez.
      -> Geralmente utilizamos a inicialização do bloco for para declararmos e inicializarmos a variável que contará o número de vezes que a expressão será executada
      -> Porém, podemos apenas declarar, ou declarar e inicializar mais de uma variável de um mesmo tipo
      -> As variáveis declaradas na inicializaçao do bloco for, só estarão disponíveis dentro do bloco for (escopo)
    -> Teste:
      -> O teste deve obrigatoriamente se uma expressão booleana
      -> É este testes (quando false) que irá determinar o término do loop for
    -> Incremento:
      -> A expressão especificada no incremento será executada a cada ciclo realizado

-> break
  -> A instrução break, quando usada em um comando for, while ou do/while, fará com que a execução do loop seja completamente interrompida
  -> Após a execução da instrução break, a execução do código seguirá após o fechamento das chaves do loop.

-> continue
  -> A instrução continue é similar às instrução break, porém, ao invés de interromper todo o loop, apenas a iteração atual será interrompida e o código prosseguirá normalmente com a próxima iteração do loop

-> labels
  -> Labels são geralmente usados quando temos loops aninhados e queremos que o loop mais externo seja interrompido ou pule para a próxima iteração dependendo de uma condição que ocorra no loop interno
  -> Exemplo

-> Capítulo 7 - Arrays
  -> Arrays são estruturas de dados capazes de representar uma coleção de variáveis de um mesmo tipo
  -> Sintaxe
  -> Todo array é uma variável do tipo reference e, por isso, precisa ser inicializada se declarada como uma variável local.
  -> Por outro lado, quando declarada como uma variável de instância, terá seu valor inicial como null;
  -> A forma mais utilizada para inicialização de arrays é por meio da palavra reservada new, seguida do tipo de elementos que o array irá armazenar, seguido de um par de colchetes com a quantidade de elementos que o array irá armazenar
  -> Ao criar um array e inicializá-lo com a palavra new, todos os seus elementos são inicializados com o valor padrão
  -> Todo array é indexado a partir de zero, portanto, ao declarar um array, de 5 posições, ele começará na posição 0 e irá até a posição 4
  -> Para atribuir valores às posições de uma array, deve-se referenciá-lo utilizando o seu índice
  -> A segunda forma de inicialização de arrays é declarar e inicializar o array em uma única instrução, colocando os valores a serem armazenados em cada posição entre chaves e separados por vírgula
  -> Para obter a quantidade de elementos em um array, usamos a propriedade length
  -> Também é válida a inicialização que utiliza o operador new e passa os valores iniciais entre chaves
  -> Arrays bidimensionais:
    -> São arrays compostos por outros arrays
    -> Sintaxe da declaração
    -> Após declarar um array bidimensional, é possível inserir valores em cada uma das posições do array e do 'subarray'
    -> Também é possível inicializar arrays bidimensionais definindo seus valores entre chaves
    -> Quando inicializamos um array bidimensional, basta que informemos apenas o tamanho do primeiro array
    -> Arrays multidimensionais

  -> Enhanced for (foreach)
    -> Este recurso foi incluído na linguagem Java a partir da versão 5
    -> Facilita bastante a navegação entre os membros de um array
    -> Exemplo