Linguagem de Programação II

Conceitos Básicos 1

Universidade do Estado do Rio de Janeiro-UERJ
Instituto de Matemática e Estatística-IME
Ciência da Computação
Professor: Alexandre Sztajnberg

Reservadas Java

- Modificadores de Acesso
  - public
  - private
  - protected
- Estruturas de controle
  - for
  - do
  - while
  - continue
  - break
  - if
  - else
  - case
  - switch
  - default
- Classes e modificadores
  - abstract
  - class
  - extends
  - final
  - implements
  - import
  - instanceof
  - interface
  - new
  - packege
  - return
  - static
  - synchronized
  - super
  - this
  - volatile
  - void
- Tipos Primitivos
  - boolean
  - int
  - byte
  - char
  - float
  - short
  - long
  - double
- Exceções
  - throw
  - throws
  - try
  - cacth
  - finnaly

Identificadores

- Identificadores ou nomes
  - Classes, objetos, atributos, métodos, variáveis e referências, constantes
- Regras para definir identificadores:
  - O primeiro caractere deve ser constituída de letra
  - O segundo em diante pode ser constituído por -, _, letras, números
  - Não pode conter espaços entre as palavras
  - Não pode usar palavras reservadas
  - A nomeação deve ser direta e intuitiva

Válidos: age_of_dog taxRateY2K HourlyEmployee ageOfDog

Inválidos (Por que?) : age# 2000TaxRate Age-Of-Dog

Camel Case

- Camel Case (ou CamelCase) é um padrão para formação de identificadores
  - Para variáveis, objetos e funções
    - a primeira palavra é totalmente escrita em minúscula
    - a segunda tem somente a sua primeira letra em maiúscula (lowerCamelCase).
    - Exemplo precoCasa, salarioFuncionarios
  - Para classes
    - somente a primeira letra de cada palavra possui a letra minúscula
    - o restante será em minúsculo(UpperCamelCase).
    - Exemplo: NomeFuncionarios, Espécies **// pode, mas não use acentos**
- O Camel Case segue todas as regras de nomenclatura citada nos identificadores.

Tipos em Java

- Primitivos
  - *Funcionam como no C, "tradicional"*
  - *Passagem por valor*
  - *Palavras reservadas*
- Inteiros
  - *byte, int, char, short, long*
- Booleanos
  - *boolean*
- Ponto flutuante
  - *float, double*
- Compostos ou referências
  - Estruturas mais complexas
  - Referências, passagem por referência
    - Ponteiros
- Array
  - <tipo>[int]
- Interface
  - interface
- Class
  - class

Espaço na Memória em Java

- Cada tipo primitivo define um espaço de memória usado e faixa de valores
- Em outros ambientes isso pode ser dependente do Sistemas Operacional e do HW
  - Ao gerar código instruções de meta-nível podem ser usadas para otimizar o uso de recursos
- Em Java este valores são definidos para a JVM e ByteCode, independente da plataforma hospedeira

Tipo Primitivo		Tamanho	Val. Padr.	Menor Valor	Maior Valor
Inteiros	byte	8 bits	0	-128	127
	short	16 bits	0	-32.768	32.767
	int	32 bits	0	-2.147.483.648	2.147.483.647
	long	64 bits	0L	-9.223.372.036.854.775.808	9.223.372.036.854.775.807

Espaço na Memória em Java

- float
  - 4 bytes
  - Armazena números fracionários. Suficiente para armazenar 6 a 7 dígitos decimais
- double
  - 8 bytes
  - Armazena números fracionários. Suficiente para armazenar 15 dígitos decimais

Tipo	Primitivo	Tamanho	Val. Padr.	Menor Valor	Maior Valor
Ponto Flutuante	float	32 bits	0.0f	-1,4E-45	3,4028E+38
	double	64 bits	0.0d	-4,9E-324	1.79769313486...E+308
Caractere	char	16 bits	'\u0000'	0	65535
Booleano	boolean	1 bit	false	false	true

Exemplos em Código

```
class Main{
  public static void main(String args[]){
    int numerodaCasa;
    boolean estaAVenda = false;
    float valorDaCasa =320.456f;

    int a, b, c=2;
  }
}
```

Tipos Compostos ou referências

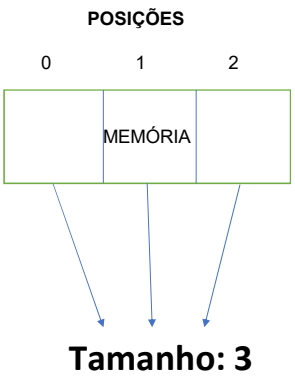
- Tipos compostos ou referências em Java são estruturas
- Dependendo do objetivo podem ser compostos por
  - Zero ou mais tipos primitivos
  - Zero ou mais métodos
  - Zero ou mais tipos compostos
    - Lembrando que são referências
  - Usados para representar objetos ou elementos do programa de forma mais completa e **encapsulada**
  - Úteis para armazenar "coleções"
- Em Java existem 3 principais tipos compostos:
  - Array
  - Classes
  - Interface

Array

- Coleção ordenada de um mesmo tipo
  - Primitivo. Valores são armazenados na memória
  - Composto. Coleção de referências à objetos ou outros arrays
- Precisam ser inicializados (new)

```
class ArrayDemo{
  public static void main(String[] args){
   int[] array; // declaração do array
   array = new int[3] ;// inicializa do array tamanho 3
   array[0] =100; //começa a contagem na posição zero
   array[1] =100;
   array[2] =100; // o tamanho do array é sempre n-1

   for (int i=0; i < array.lenght; i++)
     System.out.println ("array[" + i + "] = " + array[i]);
  }
}
```



Tamanho: 3

Classe

- Representam objetos de um mesmo tipo
- Podemos ter vários objetos instanciados a partir da mesma classe
- Podem ter campos/atributos e métodos
- Podem ser usados como uma base para criação de classes mais especializadas

```
class Bicycle {
  private int cadence = 0;
  private int speed = 0, int gear = 1;

  void changeCadence(int newValue) {cadence = newValue;}
  void changeGear(int newValue) {gear = newValue;}

  void speedUp(int increment) {
    speed = speed + increment;}

  void applyBrakes(int decrement) {
    speed = speed - decrement;
  }

  String toString() {
    String s = "cadence:" + cadence + " speed:"
               + speed + " gear:" + gear);
    return s;
  }
}
```

Interface

- Coleção de métodos, sem código, representando a interface com o "mundo exterior"
- É o primeiro passo a ser dado em um projeto de software, geralmente bem elaborado pela equipe
- Classes podem ser desenvolvidas, em várias versões, implementando a mesma interface e depois testadas
- MUITO importante e útil. Vamos depois ver que o *sort* depende disso, por exemplo ...

```
public interface Habitacao{
 public void setAbPorta(boolean abPorta);
 public boolean getAbPorta();
 public void setQtdMorad(int qtdMoradores);
 public int getQtdMorad();
 public void setAbJanela(boolean abJanela);
 public boolean getAbJanela();
}
```

```
public class Apartamento implements Habitacao{

  private boolean portaAberta, janelaAberta;
  private int numMorad;

  public void setAbPorta(boolean abPorta) {
    portaAberta = abPorta;
  }
  public boolean getAbPorta(){
    return portaAberta;
  }
  public int setQtdMorad(int qtdMoradores)
   {...}
  public int getQtdMorad() {...}

  public boolean setAbJanela(boolean abJanela)
   {...}
  public boolean getAbJanela() {...}
}
```

Declaração de variáveis

- As variáveis para serem declaradas em Java necessitam de um identificador e um tipo.
- Após sua declaração o Java não permite fazer redeclarações, ou mudanças de tipagem durante a execução do programa.
- Toda variável em Java deve ser declarada antes de ser chamada
  - Globais
    - declaradas no contexto global da classe
    - fora dos métodos
    - podem ser acessadas dentro de qualquer função dentro do programa
  - Local
    - Declaradas dentro os métodos ou recebidas por parâmetros
    - Acessadas dentro do contexto em que elas foram criadas

Declaração de variáveis

- Podem ser declaradas em qualquer ponto do código
  - Cuidado com o escopo/contexto e visibilidade
- Variáveis não podem ficar soltas
  - Sempre dentro de uma classe, ou método
  - O mesmo para métodos: não podem ficar soltos. Estamos falando de Java e Orientação à Objetos
- Se a variável é declarada localmente ou recebida por parâmetro, mas tem que ser usada "no futuro", precisa atribuir o valor ou referência para uma variável global

```
private int chave;

public void setChave (int chv) {
    chave = chv;
}
```

Declaração de Constantes

- Para declarar uma constante em Java usamos um truque
  - O modificador **final** na declaração da variável
  - Na declaração temos um atribuição literal
  - O modificador **static** pode ser usado para dar visibilidade de classe
- Para uma boa prática de programação as constantes são declaradas totalmente em maiúsculas e em sua separação usa-se underline.
- Antecipando umas dúvidas:
  - final é usado para
    - Permitir apenas uma atribuição de valor à variável de tipo primitivo
    - Não permitir herança de classe (muitas classes dos pacotes do JDK são assim)
  - static
    - ... Ou "de classe"
    - Variáveis ou métodos com este modificador são da classe e podem ser diretamente acessadas sem instanciar objetos (o que seria "de instância" ou "de objeto", que é o padrão)
    - Em outras palavras, todos os objetos desta classe compartilham esta variável ou método, seria algo como "estado coletivo"

Exemplos

```
public class Cilindro{
  static final float PI = 3.14f; // const estática
  static double diametro = 23.7 // variável estática
  final double AREA_BASE = 24.6 // constante
  double altura; // var global
                 //de instância ou de objeto

  public double volume (double alt){
    altura = alt;
    double vol = this.area() * altura;
    return vol;
  private double area(){
    double ar; //variável local
    ar = PI*(diametro/2)^2;
    return ar;
  }
  public String toString() {
    return "Cilindro. Raio: " + diametro/2 +
           "Altura: " + altura;
  }
}
```

```
public class Teste {
  public static void main (String[] a) {
    Cilindro cil;
    cil = new Cilindro ();

    double a = cil.area(); // por que?
    double v = cil.volume (10.0d);

    double al = cil.altura;
    System.out.println (cil.volume (5.0d));
    System.out.println(cil);
    System.out.println(cil.toString());
   }
}
```

Exercícios