Programação Orientada a Objetos

Turma 3

Carlos Henrique Bughi, MSc

UNIVALI
Escola Politécnica

Onde estamos?
(e para onde vamos)

- Introdução Java
  - Correção avaliação, + java
- Programar

UNIVALI

Escola Politécnica

Aula 8

Introdução a linguagem Java e a Programação Orientada a Objetos

UNIVALI

Escola Politécnica

Conceitos de OO

- Java é uma linguagem totalmente orientada a objetos, seguindo seus principais conceitos.
- A seguir serão apresentados as características da linguagem....

Variáveis: Tipos Primitivos

Tipo	Descrição
boolean	Pode assumir o valor true ou o valor false
char	Caractere em notação Unicode de 16 bits. Serve para a armazenagem de dados alfanuméricos. Também pode ser usado como um dado inteiro com valores na faixa entre 0 e 65535.
byte	Inteiro de 8 bits em notação de complemento de dois. Pode assumir valores entre $-2^7=-128$ e $2^7-1=127$.
short	Inteiro de 16 bits em notação de complemento de dois. Os valores possíveis cobrem a faixa de $-2^{-15}=-32.768$ a $2^{15}-1=32.767$
int	Inteiro de 32 bits em notação de complemento de dois. Pode assumir valores entre $-2^{31}=2.147.483.648$ e $2^{31}-1=2.147.483.647$.
long	Inteiro de 64 bits em notação de complemento de dois. Pode assumir valores entre -2^{63} e $2^{63}-1$.
float	Representa números em notação de ponto flutuante normalizada em precisão simples de 32 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável por esse tipo é 1.40239846e-46 e o maior é 3.40282347e+38
double	Representa números em notação de ponto flutuante normalizada em precisão dupla de 64 bits em conformidade com a norma IEEE 754-1985. O menor valor positivo representável é 4.94065645841246544e-324 e o maior é 1.79769313486231570e+308

Variáveis: Tipos Primitivos

```
public class TiposPrimitivos {
  public static void main( String[] args ) {
    //declara um int e atribui um valor
    int idade = 25;
    //declara um float e, depois, atribui um valor
    float valor;
    valor = 1.99f;
    //declarando um boolean
    boolean verdadeiroOuFalso = false;
    verdadeiroOuFalso = true;
    //declarando um char
    char letraA = 'A';
    letraA = 65;              //valor ASCII para 'A'
    letraA = '\u0041';        //valor Unicode para 'A'
    //declarando um byte
    byte b = 127;
    //declarando um short
    short s = 1024;
    //declarando um long
    long l = 1234567890;
    //declarando um double
    double d = 100.0;
    //declaração múltipla
    int var1=0, var2=1, var3=2, var4;
  }
}
```

Variáveis: Tipos Referências

- As variáveis do tipo de referência são aquelas constituídas de classes, vetores ou interfaces específicas (Objetos). Por conta disso o seu uso requer cuidados especiais, sobretudo em rotinas de comparação.
  - Principais tipos de referência são
    - String
    - Array
    - Integer
    - Quaisquer outras classes que for criada dentro de um programa

Classe String

- Classe String
  - Uma das classes mais importantes para a linguagem java;
  - Com ela é possível escrever objetos que recebam conjuntos de caracteres;
  - Exemplo:
    - String a = “Java é legal”;
  - Apesar de não ser um tipo primitivo, a classe String se comporta como um, ou seja, não é necessário criar instância, basta declara-la e usa-la;
  - Possui diversos métodos para a manipulação dos valores que são recebidos;

Classe String

- Classe String, métodos:
  - charAt(pos) – devolve o char da posição;
  - concat(Str) – Concatena Str com o objeto e retorna;
  - equals(Str) – Compara se duas Str são idênticas;
  - equalsIgnoreCase(Str) – Compara se duas Str são iguais;
  - length() – Tamanho do Objeto;
  - indexOf(ch) – retorna a posição do char;
  - lastIndexOf(ch) – retorna a última posição do char;
  - replace(ch1,ch2) – Troca de caracteres;
  - startsWith(str) – Retorna true com Str no início;
  - endsWith(str) – Retorna true com Str no final;
  - Substring(inicio,fim) – Retorna a subcadeia;
  - toLowerCase() – Retorna a String em minúsculo;
  - toUpperCase() – Retorna a String em maiusculo;
  - trim() – Retira os espaços em branco;

Classe String

```
String str = "Isto é uma String do Java";

// O método split quebra a String e várias outras,
// pelo separador desejado
String[] palavras = str.split(" ");

int i = str.indexOf("uma"); //retorna o índice da palavra na String

if( str.startsWith("Olá") || str.endsWith("Mundo!") ) {
  // testa o começo e o fim da String - retorna boolean
}

str = str.trim();  // elimina os espaços em branco no início e fim

str = str.replace('a','@');  // substitui os caracteres

// substitui uma palavra (usa expressões regulares)
str = str.replaceAll("String","Cadeia de caracteres");
```

Classe String

- Atenção ao código:

```
String a = “java”;
String b = “java”;
if (a == b) {
    System.out.println(“a é igual a b”);
} else {
    System.out.println(“a e b são diferentes”);
}
```

Resultado: a e b são diferentes

Classe String

- Para comparar se duas Strings são iguais deve-se utilizar o método equals() ou equalsIgnoreCase();
- Ao utilizar o operador == , é comparado se os objetos são iguais (compartilham a mesma posição de memória).

```
String a = “java”;
String b = “java”;
if (a.equals(b)) {
        System.out.println(“a é igual a b”);
} else {
        System.out.println(“a e b são diferentes”);
}
```

Resultado: a é igual a b

Vetores e matrizes

- Relembrando
  - Vetores são estruturas utilizadas para armazenar um conjunto de dados.
  - Em java:
    - Todos os dados armazenados devem ser do mesmo tipo de dados.
    - Por exemplo, todos os valores podem ser inteiros (tipo int).
    - Podemos definir vetores de tipos primitivos ou de referência;

Vetores tipo primitivo

- Sintaxe
  - tipoArray [ ] nome = new tipoArray[numero]
    - tipoArray: tipo do array seguido de [ ]. Ex. String[], int[], char[];
    - Nome: nome do array.
    - new tipoArray[numero]: instancia um objeto da classe Array (pacote java.lang) com “numero” elementos;
  - Exemplo:
    - int[] ficha = new int[3]; // cria array
    - ficha[0] = 10; // atribui valor 10 no índice 0
    - ficha[1] = 12; // atribui valor 12 no índice 1
    - ficha[2] = 3; // atribui valor 3 no índice 2
    - System.out.println(ficha.length) // imprime o tamanho do array

Vetores

- Sintaxe
  - tipoArray [ ] nome = {val1,..., valn};
    - Outro modo de criação de um array.
    - Cada valor passado assumirá automaticamente uma determinada posição;
  - Exemplo:
    - int[] ficha = {10,12,3}; // cria array atribuindo os valores 10, 12 e 3
    - System.out.println(ficha.length) // imprime o tamanho do array

Matrizes

- Para utilizar array bidimensionais, bastar agregar mais um conjunto de [ ] na sua estrutura, do seguinte modo:
  - int [] [] armario = new int[2] [3];
  - armario[0][0] = 10;
  - armario[0][1] = 12;
  - armario[0][2] = 5;
  - armario[1][0] = 14;
  - armario[1][1] = 4;
  - armario[1][2] = 1;

Matrizes

- Ou ainda
  - int [] [] armario ={ {10,12,5} , {14,4,1} };

- Obtendo o tamanho;
  - armario.length; //qtde linhas
  - armario[0].length; //qtde de colunas

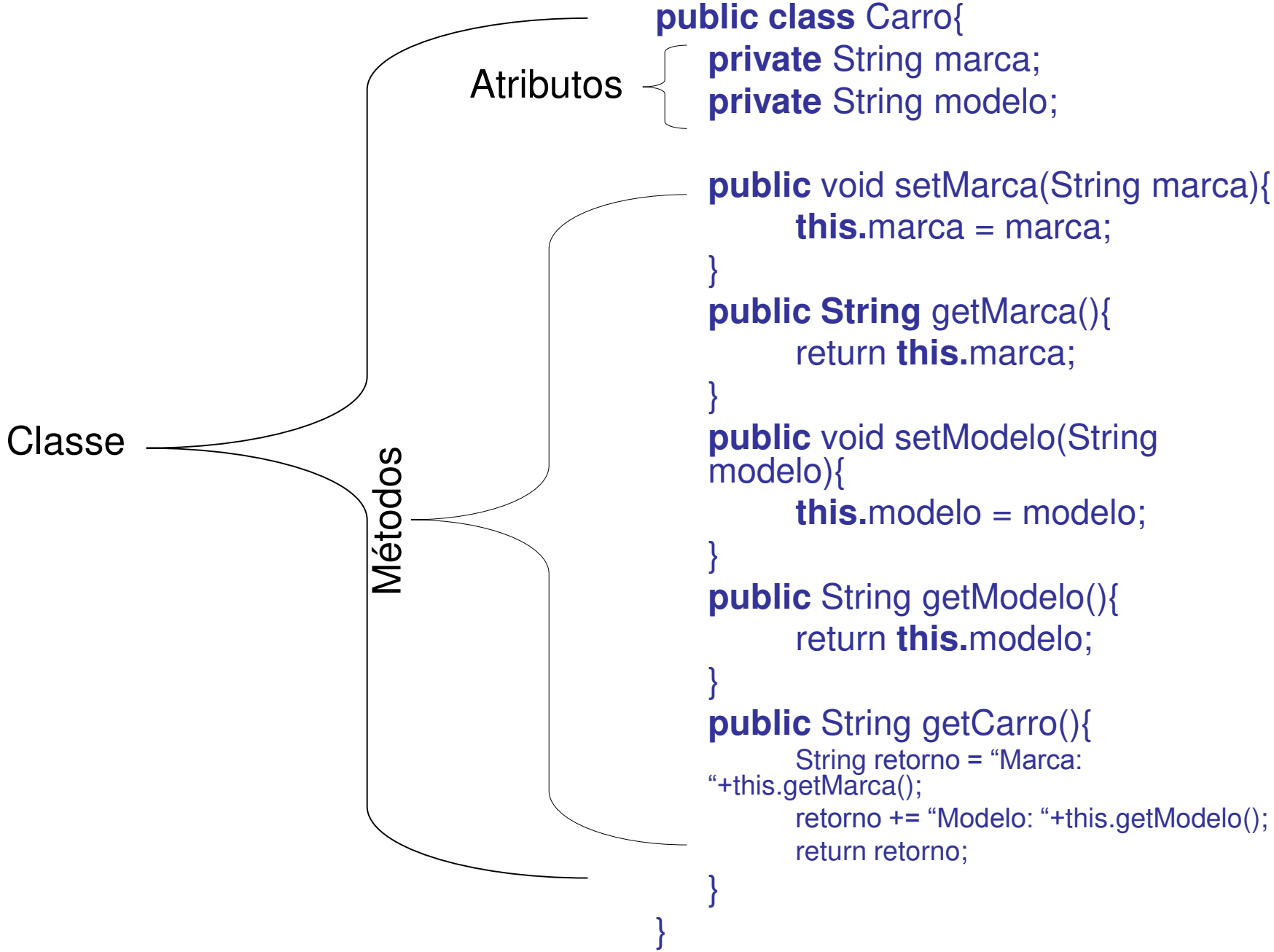
Vetores de objetos

- Além dos tipos primitivos, a linguagem java também permite a criação de arrays de objetos;
- Supondo a classe Ficha com o construtor Ficha(nome, salario), tem-se a definição de um array de ficha:
  - Ficha[] ficha = new Ficha[2];
  - A grande diferença para o tipo primitivo é que cada elemento do array deve ser instanciado, ou seja:
    - ficha[0] = new Ficha(“joao”,1500);
    - ficha[1] = new Ficha(“Zezinho”, 3000);

Criando classes

- Uma classe no Java deve ser definida da seguinte forma:

```
visibilidade class nome_da_classe{
        Atributos
        Métodos (operações)
        Construtores
}
```


Classe
Atributos
Métodos
public class Carro{
private String marca;
private String modelo;
public void setMarca(String marca){
this.marca = marca;
}
public String getMarca(){
return this.marca;
}
public void setModelo(String modelo){
this.modelo = modelo;
}
public String getModelo(){
return this.modelo;
}
public String getCarro(){
String retorno = "Marca: "+this.getMarca();
retorno += "Modelo: "+this.getModelo();
return retorno;
}
}

Criando classes

- Neste exemplo definimos a classe carro com dois atributos: marca e modelo e os métodos para modificar e recuperar os valores dos atributos; Além do método que retorna o valor de todos os atributos do objeto

- Dentro dos métodos da classe utilizamos a variável this para referenciar o próprio objeto, e este deve ser o procedimento quando queremos referenciar o objeto em seus métodos;

Criando classes

Regras para Nomeação de Classes

Para se nomear classes, existem alguns requisitos que devem ser observados:

- Classes são nomeadas com um substantivo no singular
- O nome de uma classe deve iniciar com a **primeira letra maiúscula;**
- Não devem ser utilizados símbolos de sublinhado ("_") - nomes compostos por múltiplas palavras são organizados com todas as palavras juntas, onde a primeira letra de cada uma fica em maiúscula.

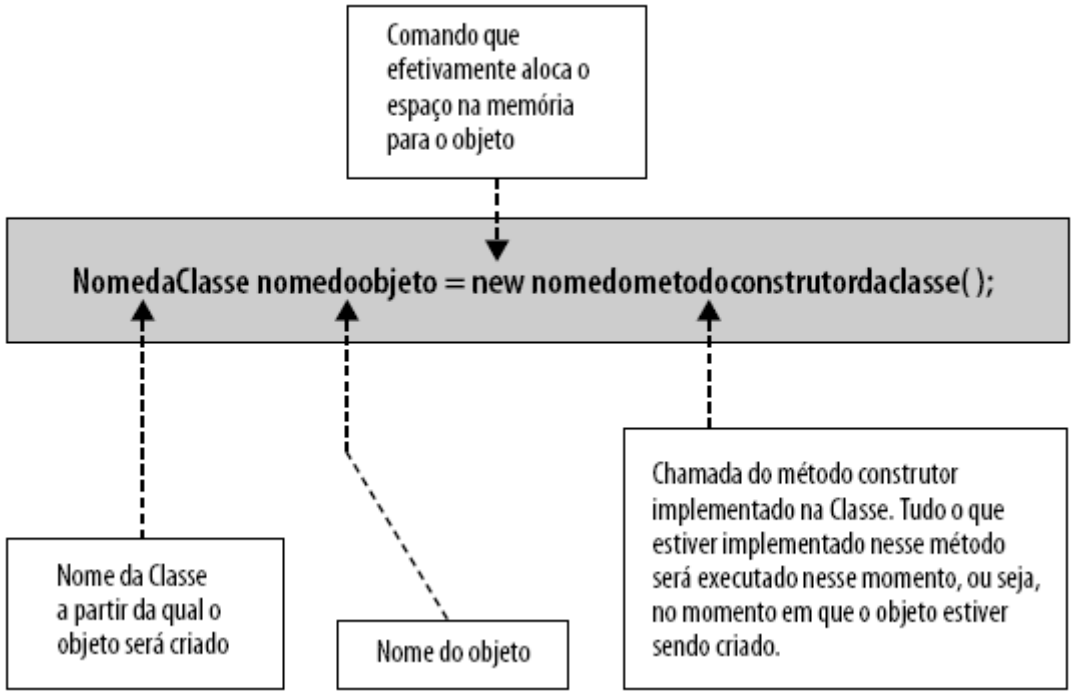
Exemplos: Aluno, Professor, FolhaPagamento.

Regras para Definição de Atributos

- Normalmente, atributos são colocados no início da definição da classe, depois do primeiro { . Também pode ser colocados bem no final, antes do último }.
- Devem começar com letra minúscula.

Instanciando Classes

- Para instanciar (criar um objeto a partir de uma classe existente) uma classe, devemos utilizar a instrução **new** seguida do nome da classe;



Instanciando Classes

- Uma vez instanciada, podemos acessar os atributos e métodos públicos da classe;
- Devemos utilizar o operador . (ponto) após o nome do objeto que instancia a classe;

```
Carro carro = new Carro();
carro.setMarca("Ford");
carro.setModelo("Focus");
System.out.println(carro.getCarro());
System.out.println(carro.getMarca());
System.out.println(carro.marca); //ERRO
```

Visibilidade de atributos e métodos

- Os métodos e atributos de uma classe podem ser definidos como: privados (private), públicos (public) e protegidos (protected) ;
- Cada um dos tipos define como o atributo ou método se comporta perante a classe, subclasses e o restante do sistema.

Visibilidade de atributos e métodos

- **Public**
  - Um atributo ou método definido como public torna-o acessível em qualquer lugar da classe, de suas subclasses, bem como por qualquer outra classe que a utiliza.

```
public class Carro{
        public String modelo;
        public void setModelo(modelo){
                this.modelo = modelo;
        }
}
....

Carro carro = new Carro();
carro.setModelo(“Ford”);
carro.modelo = “Chevrolet”;
```

Visibilidade de atributos e métodos

- **Public**
  - Neste exemplo o atributo modelo foi definido como público, assim como o método setModelo. Desta forma é possível acessar o atributo tanto dentro da classe quanto fora dela;
  - **Encapsulamento**: O mais indicado é termos os atributos definidos como private ou protected e métodos get (obter) e set (atribuir), evitando acesso direto ao seu conteúdo.

Visibilidade de atributos e métodos

- **Private**
  - Os atributos e métodos definidos como private são visíveis somente na classe que os criou, ou seja, subclasses ou da classe que contêm a classe não podem acessar os atributos ou métodos.

```
public class Pessoa{
      private String tipo;
      protected String nome;
      protected String endereco;
      protected void setTipo(tipo){
         this.tipo = tipo;
      }
  }
public  class Estudante extends Pessoa{
     protected String curso;
     public Estudante() {
         super.setTipo("E");
     }
  }

....
  Estudante estudante = new Estudante();
```

Visibilidade de atributos e métodos

- **Protected**
  - Atributos ou métodos definidos como protected são visíveis pela classe que os criou e por suas subclasses (classes que herdam a classe principal)
  - Podem ser acessados pela classe filha através da variável **super**.

```
public class Veiculo{
        protected String modelo;
        public void setModelo(modelo){
            this.modelo = modelo;
        }
}
public class Caminhao extends Veiculo{
        protected int eixos;
        public void setModelo(modelo){
            super.modelo = modelo;
        }
        public void setEixos(eixos){
            this.eixos = eixos;
        }
}
```

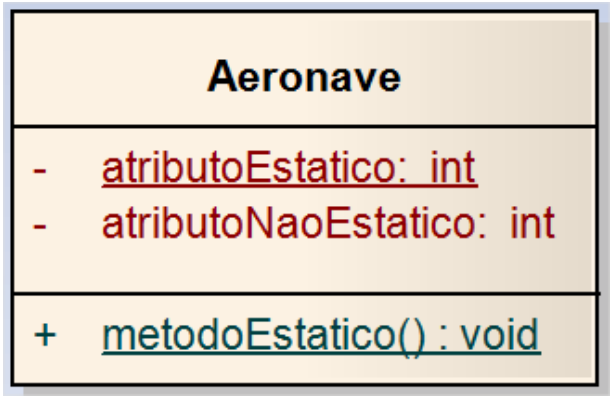
```
Veiculo carro = new Veiculo();
carro.setModelo("Ford");
carro.modelo = "Chevrolet"; //ERRO

Caminhao caminhao = new Caminhao();
caminhao.setModelo("Scania");
caminhao.setEixos(8);
```

Nível de classe e Nível de instância

- Um **atributo** ou **campo** é uma variável declarada no corpo de uma classe. Ele serve para armazenar o **estado de um objeto** (e neste caso é chamado de atributo de instância) ou o **estado de uma classe** (atributo de classe).

Nível de classe e Nível de instância

- Atributos de Classes:
  - Um atributo de classe é um atributo declarado com o modificador **static**. Os atributos de classe são associados com a classe e não com os objetos (ou instâncias) criados a partir dela. Isso significa que quando um atributo de classe é modificado, todos os objetos criados a partir da mesma classe enxergam a alteração



  - Sublinhados no diagrama de classes
  - Podem ser acessados sem a necessidade de criar um objeto
  - Atributos da classe e não do objeto em específico

```
class Empregado{

    static int numeroDeEmpregados; // atributo de classe

    public int cargo;              // atributo de instância (objeto)

}
```

Nível de classe e Nível de instância

- Atributos de Instâncias:
  - Um atributo de objeto (também conhecido por atributo de instância) é uma variável declarada sem o modificador **static**. Os atributos de instância pertencem (são associados) aos objetos e não às classes. Quando um atributo de objeto é modificado, somente o objeto a que ele pertence enxerga a alteração (isso porque cada objeto possui o seu). Um atributo de objeto (ou instância) é criado quando o objeto é criado (comando **new**) e é destruído quando o objeto a que pertence for destruído.

```
class Empregado{

   public int cargo;

   public void contrataComoProgramador(void){
      cargo = 2; // Dois é o código de programador
   }

}
```

Nível de classe e Nível de instância

- Métodos de Classes:
  - Um método de classe é um método que acessa somente os **atributos de classe** definidos na sua própria classe. Para declarar um método deste tipo, especifique a palavra-chave **static:**

```
class Empregado {

   private static int numeroDeEmpregados;

   static void ajustaNumeroDeEmpregados(int numero){
      numeroDeEmpregados = numero;
   }

}
```

Nível de classe e Nível de instância

- Não confunda ESTÁTICO com CONSTANTE!!!!

Constantes

- Uma **constante** é uma variável que só permite a leitura do seu conteúdo (valor). Uma vez definida, uma constante não pode mais ser modificada. As constantes são declaradas através da palavra-chave **final**. As constantes também podem ser de classe ou de objeto. Por questões de performance, sempre crie constantes de classe, como no exemplo:

```
class ProgramaExemplo4 {

  final static TOTAL = 10;

   public static void main(String parametros[]){

      int contador = 0;
      while(contador < TOTAL) contador++;

    }
}
```

Construtores e Destrutores

- O construtor é uma função definida na classe e é executada sempre que o objeto é criado (a classe é instanciada);
- Java não possui um destrutor explícito. Um objeto é retirado da memória automaticamente através de um “coletor de lixo” sempre que não existir mais nenhuma referencia ao objeto.

Construtor

- Um construtor em java consiste em um método público possuindo o mesmo nome da classe;
- O construtor pode ser construído de forma que aceite parâmetros na sua definição;

```
public class Veiculo{
    private String nome;

    public Veiculo(){
        this.nome = “Genérico”; //inicia o atributo nome
    }
}

Veiculo veiculo1 = new Veiculo();
Veiculo veiculo2 = new Veiculo();
```

Pacotes

- A linguagem java permite reunir classes de um determinado propósito em pacotes (package);
- Um pacote nada mais é que o diretório onde as classes se encontram;
- É utilizado para definir o domínio de um conjunto de classes, permitindo a diferenciação de classes com o mesmo nome;
- Por padrão, o nome do pacote é composto por uma URI (Universal Resource Identifier, ou Identificador Universal de Recursos) reversa;
- Atributos e métodos de uma classe definidos como **protected** são visíveis por todas as classes do pacote;

Pacotes

- A definição de onde uma classe se encontra é feita utilizando a palavra package;
  - Ex: supondo que a classe Retangulo esteja na pasta br\univali\cc\prog\geometria, a primeira instrução da classe deverá ser:
    - package br.univali.cc.prog.geometria;

- Para utilizar uma classe que se encontra em um outro pacote, deve-se importá-la através do comando import:
  - Ex: import br.univali.cc.prog.geometria.Retangulo;

Legalizing Package Names

Domain Name	Package Name Prefix
clipart-open.org	org.clipart_open
free.fonts.int	int_.fonts.free
poetry.7days.com	com._7days.poetry

Pacotes

- É possível ainda importar todas as classes de um pacote, utilizando o asterisco.
  - Ex: import br.univali.cc.prog.geometria.*;

- Atenção!
  - O uso do asterisco não importa as classes existentes nos subpacotes.
  - Por conveniência, o compilador Java automaticamente importa três pacotes por arquivo:
    - Pacotes sem nome;
    - O pacote java.lang;
    - O pacote atual;

Lendo valores a partir do console

- A partir da versão 1.5 é possível utilizar a classe java.util.Scanner para recuperar valores primitivos e strings;
- A classe Scanner quebra uma entrada (input) usando como delimitador o espaço em branco “ “.
- Os resultados podem ser convertidos em valores de diferentes tipos usando os métodos nextTipo() existentes;

Lendo valores a partir do console

- Exemplo de uso:
  - Lendo String

```
 1 import java.util.Scanner;
 2
 3 public class Leitura
 4 {
 5       public static void main(String[] args)
 6       {
 7          Scanner input = new Scanner(System.in);
 8          System.out.print("digite um valor:");
 9          String valor = input.nextLine();
10          System.out.println("A valor digitado foi "+valor);
11       }
12 }
```

Lendo valores a partir do console

- Exemplo de uso:
  - Lendo int

```
1  import java.util.Scanner;
2
3  public class Leitura
4  {
5        public static void main(String[] args)
6        {
7           Scanner input = new Scanner(System.in);
8           System.out.print("digite um valor:");
9           int valor = input.nextInt();
10          System.out.println("A valor digitado foi "+valor);
11       }
12 }
```

Classe Wrapper

- Existem classes especiais para cada um dos tipos primitivos;
- Sua função é dotar cada tipo com métodos para que possamos resolver problemas do tipo:
  - Como converter um objeto String para o tipo primitivo int;
- O principal método de utilização dos numéricos é o **parseTipo**. Então, convertendo uma String para os diversos tipos, por exemplo, tem-se:
  - byte a = Byte.parseByte(“1”);
  - short b = Short.parseShort(“1”);
  - int c = Integer.parseInt(“1”);
  - long d = Long.parseLong(“1”);
  - float e = Float.parseFloat(“1”);
  - double f = Double.parseDouble(“1”);

Classe Wrapper

- Para converter uma String para o tipo boolean, utiliza-se o método:
  - Boolean.getBoolean(str);
- Para converter uma String para o tipo char, utiliza-se um método da própria classe String;
  - “oi”.charAt(0) // devolve o caracter o