

Classes e Objetos em Java

Cleyton Caetano de Souza

IFPB – Campus Monteiro

cleyton.caetano.souza@gmail.com

Roteiro

- Introdução
- Classes e Objetos
- Construtores
- A classe Object
- Métodos
- Atributos e Variáveis
- Modificadores de Acesso

Classes

- O que é uma classe?

Sintaxe para Declaração de uma Classe

Modificador
de Acesso

class

Nome da
Classe

{

}

Exemplos de Classe

```
public class Professor {  
  
}
```

```
public class Aluno {  
  
}
```

```
public class Disciplina {  
  
}
```

```
public class Curso {  
  
}
```

Nomes de Classe válidos

1. Não pode ser uma palavra reservada
2. Não pode conter espaços em branco
3. Os nomes de **Classes** devem começar com
 1. uma letra
 2. um cifrão: \$
 3. um underline: _
4. O restante do nome pode ser composto por letras, números, cifrão ou underline
5. Java é *case sensitive*

Convenções para Nomenclatura de Classes

- Os nomes de **Classes**, geralmente, são substantivos
- A primeira letra do nome de uma **Classe** SEMPRE deve ser Maiúscula.
- Se o nome for composto por diversas palavras, a primeira letra de cada palavra deve ser maiúscula enquanto que o restante das letras devem ser minúsculas
 - Seguindo o padrão “*Camel Case*”
- No geral, não se utiliza números, cifrão ou *underline* para nomes de classes

Exercício

- Caracterize os nomes de **classe** como (1) válido ou não válido e (2) seguindo ou não as convenções de nomenclatura de classes que aprendemos

a) cliente
b) 12asd
c) \$_
d) Cliente_Credito
e) _\$
f) \$1
g) Aluno.Medio
h) X9

i) Bytes
j) New
k) float
l) Inteiro
m) A
n) CEP
o) \$int
p) Aluno

q) Programação
r) _Professor
s) BigLong
t) Blob
u) :Esposa:
v) #Disciplina
w) string
x) isso_não_é_legal

Classes


- Do que uma classe é composta?
 - ATRIBUTOS
 - São as características que o **objeto** vai ter!
 - MÉTODOS
 - São as ações que o **objeto** vai poder realizar

O que é um objeto?

- Um objeto possui estado e comportamento, os quais são descritos pela classe.

O que é um objeto?

- A variável 'professor' é um objeto?



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

O que é um objeto?

- A variável professor é um objeto?
 - Resposta: Não. A variável professor me permite acessar o objeto que está na memória.

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

O que é um objeto?

- Como é feita a associação entre a variável professor e o objeto na memória?

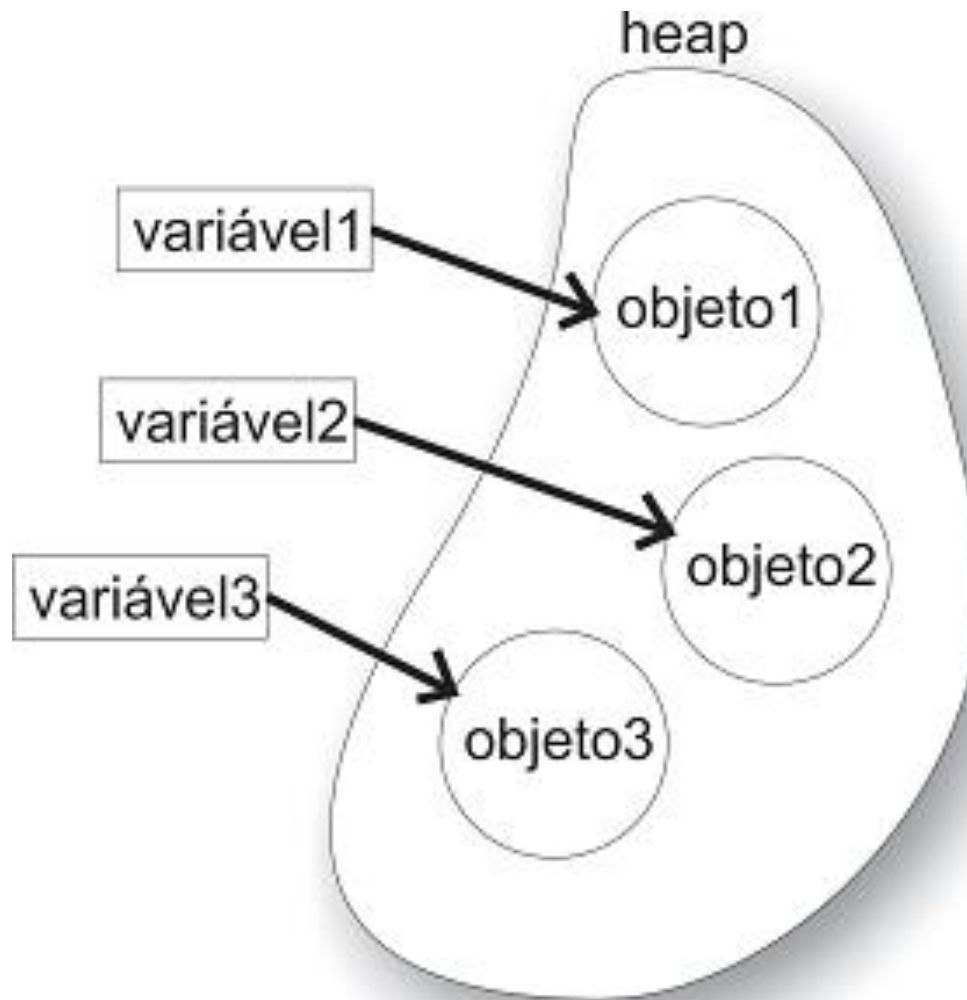
```
public static void main(String[] args) {  
    Professor professor;  
  
}
```

O que é um objeto?

- Como é feita a associação entre a variável professor e o objeto na memória?
 - Resposta: Através do operador **new**. O operador **new** constrói (instancia) o objeto na memória.

```
public static void main(String[] args) {  
    Professor professor = new Professor();  
}
```

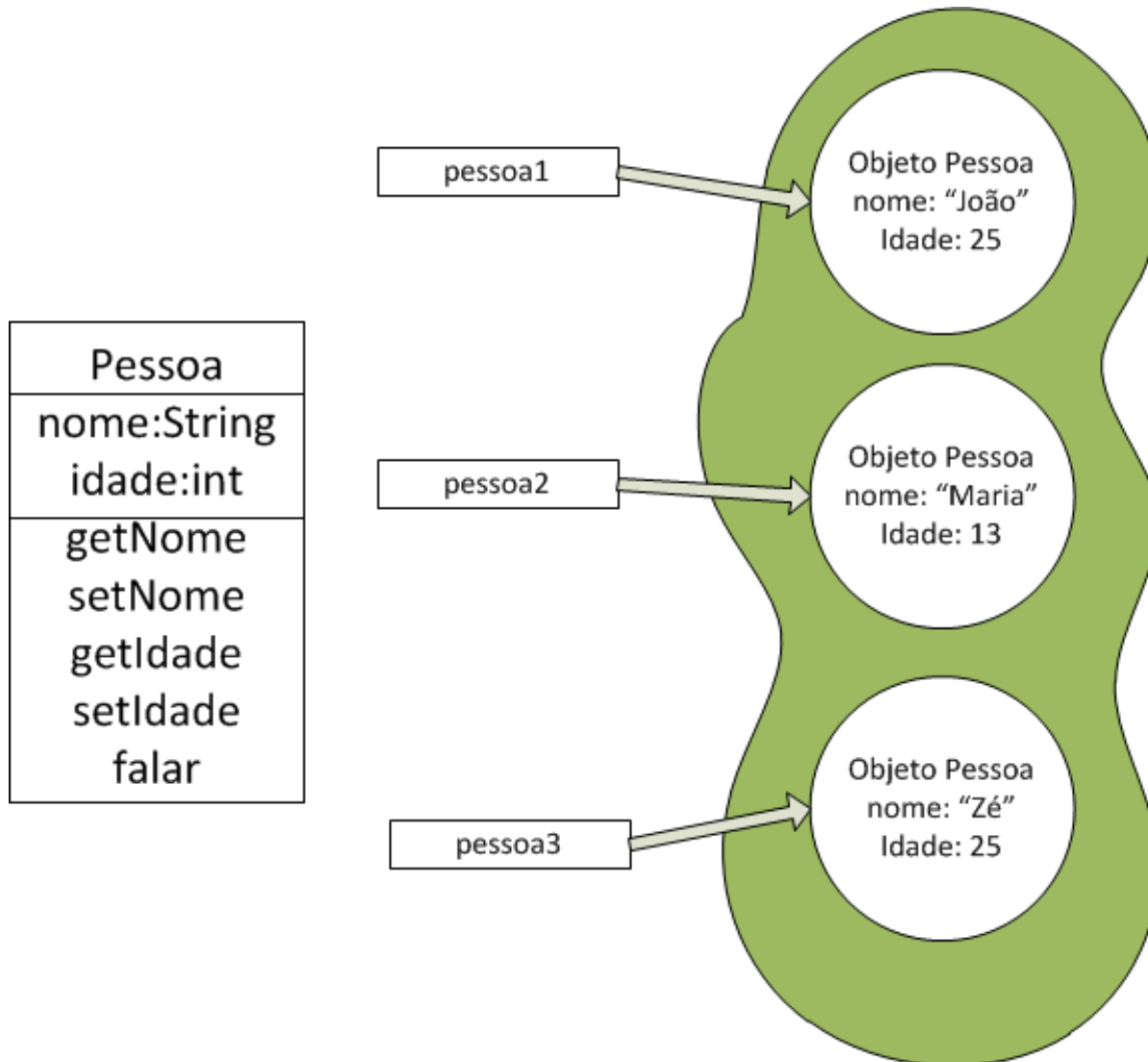
Como funciona na memória



Os atributos de um Objeto

- Todos os objetos de uma mesma classe tem os mesmos atributos, mas objetos diferentes podem ter valores diferentes para o mesmo atributo
- Da mesma forma como no mundo real, dois objetos tem os mesmos descritores, mas valores diferentes para esses descritores
 - Ex. No mundo real, todas as Pessoas possuem um atributo chamado altura, mas cada pessoa possui seu valor de altura.

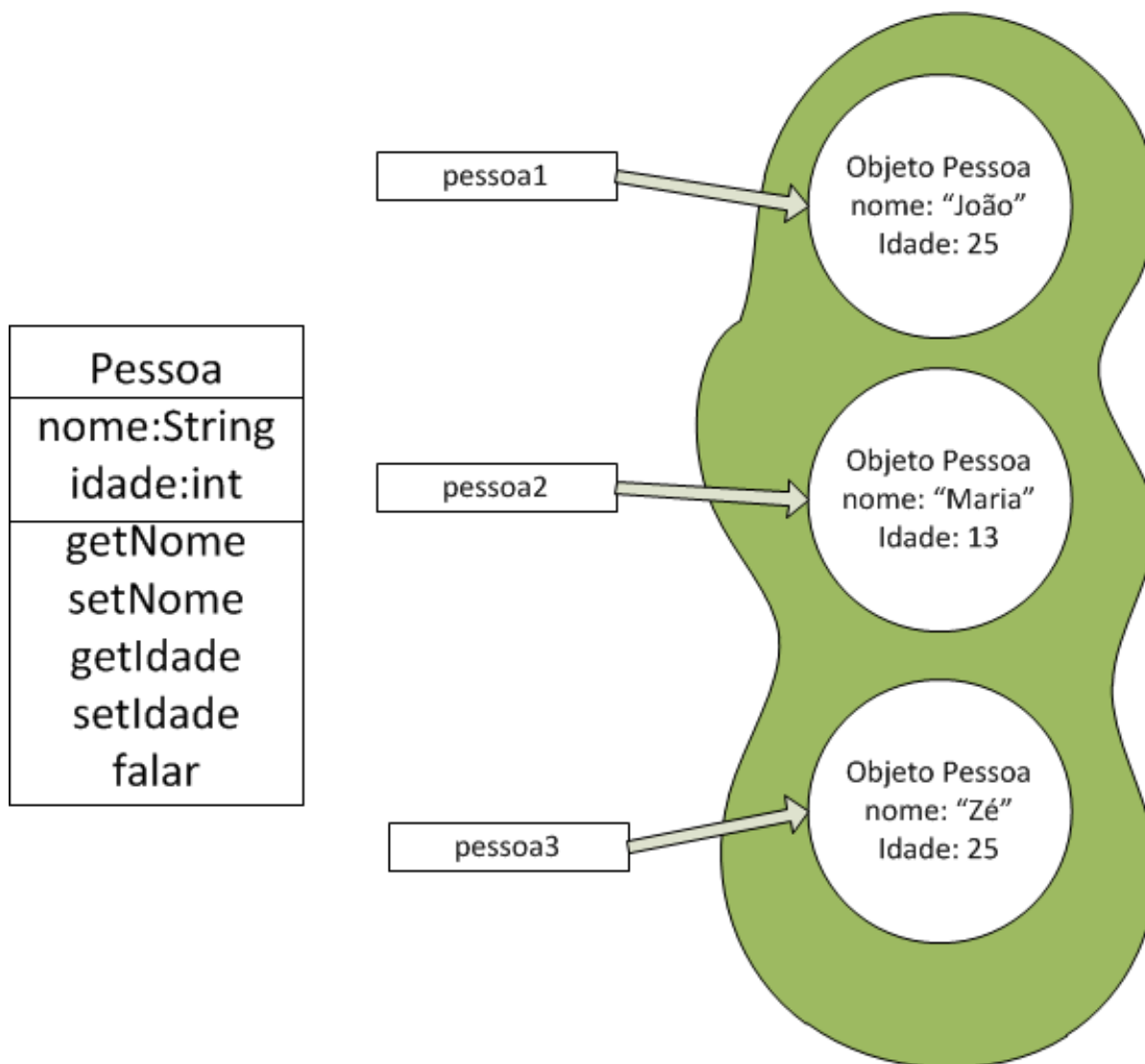
Qual o valor da propriedade nome do objeto apontado pela variável pessoa2?



Os métodos de um Objeto

- Ao invocar o método de um objeto (por meio de sua variável de referência, eu estou requisitando que o código que há dentro daquele método seja executado
 - Como se fosse uma função
- Se o método altera ou faz uso de alguma das propriedades descritas na Classe são os valores das propriedades do objeto que está executando o método que serão alteradas e/ou acessadas

Qual o valor retornado por pessoa2.getNome()?



Variáveis de Referência

- Como a variável 'professor' “aponta” para o objeto do tipo 'Professor' na memória, ela é chamada de variável de referência

```
public static void main(String[] args) {  
    Professor professor = new Professor();  
}
```



Variáveis de Referência

- Através da variável de referência, eu posso acessar os atributos e métodos **públicos** do objeto utilizando o operador ‘.’ (ponto)
- Isso é diferente do que acontece com variáveis de tipos primitivos (byte, int, long, float, double, char e boolean).

Criando Objetos

- Construtores!
 - As classes possuem por padrão um “método” construtor que é utilizado para instanciar (i.e., criar) objetos!
 - Esse construtor tem SEMPRE o mesmo nome da classe e é invocado utilizando a palavra reservada **new**

Construtores

- Entendendo cada elemento

```
Pessoa pessoa = new Pessoa();
```

- Para que servem construtores?

Construtores

- Entendendo cada elemento

```
Pessoa pessoa = new Pessoa();
```

- Para que servem construtores?
 - Servem para instanciar (criar) objetos e associar esses objetos às variáveis de Referência

Construtores

- Entendendo cada elemento

```
Pessoa pessoa = new Pessoa();
```

- Qual o valor da variável de referência, antes de instanciar o objeto?

Construtores

- O valor padrão da variável de referência que não está apontando para nenhum objeto é **null**

```
public class Pessoa {  
    private Celular celular;  
  
    public void printCelular() {  
        System.out.println(celular);  
  
        celular = new Celular();  
  
        System.out.println(celular);  
    }  
}
```

Console

<terminated> Principa

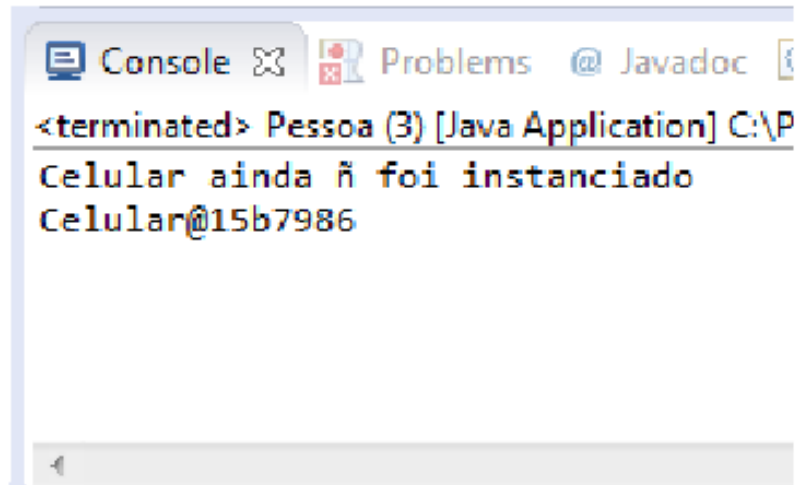
null

Celular@186d4c1

A palavra reservada **null**

- É o valor padrão de uma variável de referência
 - Ou seja, antes de ser inicializada com o endereço para o objeto, o valor da variável de referência é **null**

```
public class Pessoa {  
    private Celular celular;  
  
    public void printCelular() {  
        if (celular == null)  
            System.out.println("Celular ainda ã foi instanciado");  
  
        celular = new Celular();  
  
        System.out.println(celular);  
    }  
}
```



The screenshot shows an IDE console window with tabs for 'Console', 'Problems', and 'Javadoc'. The console output is as follows:

```
<terminated> Pessoa (3) [Java Application] C:\P  
Celular ainda ã foi instanciado  
Celular@15b7986
```

Construtores

- Também servem para inicializar o estado de alguma propriedade do objeto

```
Scanner leitor = new Scanner(System.in);
```

- Os construtores de algumas classes das **bibliotecas padrão de Java** podem requerer um parâmetro de entrada
- Como implementar construtores? Como implementar construtores que recebam parâmetros de entrada?

Construtores

- Onde está o método construtor da classe Pessoa?

```
public class Pessoa {
```

Métodos “Implícitos”

- Resposta: Ele não está explicitado, mas existe dentro do código da classe.
 - Na verdade, **o método construtor não é o único método que não está sendo apresentado no código da classe**
 - Toda classe em Java possui por padrão 11 métodos que ela herda da classe Object
- <http://docs.oracle.com/javase/6/docs/api/java/lang/Object.html>

A classe Object

- Em Java todas as Classes que já existem e todas que você vai criar vão herdar POR PADRÃO da classe Object
 - Herdar da classe Object significa que elas vão ter os mesmo métodos e atributos que a classe Object tem
 - Principais métodos da classe Object que vocês vão usar frequentemente
 - Método toString
 - Método equals
 - Método hashCode

Reescrevendo o método construtor

O construtor deve ser PUBLIC para que seja possível usá-lo

```
public class Pessoa {
```

```
    public Pessoa() {
```

```
        /*
```

```
        aqui dentro eu posso  
        escrever o código  
        que eu quero que seja  
        executado sempre que  
        um novo objeto do tipo  
        Pessoa for criado
```

```
        */
```

```
    }
```

```
}
```

Esse é o construtor da classe Pessoa

O construtor deve ter o MESMO nome da classe

O construtor não tem NENHUM tipo de retorno

Reescrevendo o método construtor

```
public class Pessoa {  
  
    public Pessoa() {  
        System.out.println("Uma nova pessoa foi criada!");  
    }  
  
    public static void main(String[] args) {  
  
        Pessoa p = new Pessoa();  
  
    }  
}
```

Construtor que imprime uma mensagem no console cada vez que um novo objeto do tipo Pessoa é instanciado

Console Problems Javadoc Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (06/11/20:
Uma nova pessoa foi criada!

Reescrevendo o método construtor

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa() {  
        nome = "desconhecido";  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public static void main(String[] args) {  
        Pessoa p = new Pessoa();  
        System.out.println(p.getNome());  
    }  
}
```

Construtor que inicia o **nome** da pessoa com o valor "desconhecido"

Console Problems Javadoc Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe
desconhecido

Construtores

- O Construtor default não tem parâmetros
 - Exemplo

```
Pessoa pessoa = new Pessoa();
```

- Como criar construtores que requerem parâmetros de entrada?

Escrevendo um construtor com Parâmetros

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeDaPessoa) {  
        this.nome = nomeDaPessoa;  
    }  
  
    public static void main(String[] args) {  
  
        Pessoa p = new Pessoa("Cleyton");  
  
    }  
}
```

Esse construtor recebe como parâmetro o valor que deve ser atribuído ao atributo **nome**

Sobrecarga de Construtores

- Uma classe pode ter mais que um construtor
- Isso é chamado de **sobrecarga de construtores**
- Cada construtor da classe deve ter uma assinatura única, para poder diferenciá-lo

Uma classe com múltiplos construtores

Uma classe pode ter múltiplos construtores

```
public class Pessoa {  
    private String nome;
```

```
    public Pessoa() {  
        System.out.println("Uma nova pessoa foi criada");  
    }
```

```
    public Pessoa(String nomeDaPessoa) {  
        nome = nomeDaPessoa;  
    }
```

```
}
```




Isso significa que haverá mais de uma forma disponível para instanciar o objeto

Sobrecarga

- O que é sobrecarga em OO?
 - **Sobrecarga permite a existência de vários métodos de mesmo nome, porém com assinaturas diferentes.**
 - Fica a cargo do compilador escolher de acordo com as listas de argumentos os métodos a serem executados.
- Além da Sobrecarga de Construtores é possível também Sobrecarregar Métodos

Sobrecarga de Métodos

```
public class Pessoa {  
  
    public String dizerOla() {  
        return "Olá";  
    }  
  
    public String dizerOla(String nome) {  
        return "Olá " + nome;  
    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa();  
        System.out.println(pessoa.dizerOla());  
        System.out.println(pessoa.dizerOla("Cleyton"));  
    }  
}
```

 Console  Problems @ Javadoc  Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)

Olá

Olá Cleyton

Sobrescrita de Métodos

- A sobrescrita é um conceito complementar a sobrecarga
- **A sobrescrita está diretamente relacionada à herança.**
 - Com a sobrescrita é possível alterar métodos herdados
- A sobrescrita de métodos consiste basicamente em criar um novo método na classe filha contendo a mesma assinatura e mesmo tipo de retorno do método da classe mãe
- Que métodos na classe pessoa poderíamos sobrescrever?

toString()

- O método toString() retorna uma representação, na forma de String, do objeto
 - Por padrão o método toString retornará o valor da referência
- A JVM entenderá que essas duas linhas querem dizer a mesma coisa, que é imprimir uma representação por escrito do objeto apontado por pessoa

```
System.out.println(pessoa);
```

```
System.out.println(pessoa.toString());
```

Utilizando o toString() padrão

```
public class Pessoa {  
  
    private String nome;  
  
    public Pessoa(String nomeDaPessoa) {  
        nome = nomeDaPessoa;  
    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa("Cleyton");  
  
        System.out.println(pessoa);  
  
    }  
}
```

Essa linha está chamando o método toString()

System.out.println(pessoa);

Aqui, a saída do método toString() padrão

Console Problems Javadoc Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\Java\jre7

Pessoa@f62373

Sobrescrevendo o toString()

Sobrescrevendo o método toString()

```
public class Pessoa {  
    private String nome;  
  
    public Pessoa(String nomeDaPessoa) {  
        nome = nomeDaPessoa;  
    }  
  
    public String toString() {  
        return "O nome da pessoa é " + nome;  
    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa("Cleyton");  
        System.out.println(pessoa);  
    }  
}
```

O método sobrescrito tem a exata mesma assinatura do método original

Aqui, a saída do método toString() que foi sobrescrito

Console Problems @ Javadoc Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\Ja
O nome da pessoa é Cleyton

equals()

- O método equals serve para comparar se dois objetos são iguais
 - Ele retorna **true** se os objetos são iguais e **false** se eles forem diferentes.
- Em Java, comparar dois objetos utilizando o operador == testa se ambas as variáveis apontam para o mesmo objeto.
 - O método equals padrão faz a mesma coisa

Utilizando o equals() padrão

Comparando duas variáveis de referência que apontam para o mesmo objeto

Comparando duas variáveis de referência que apontam objetos diferentes

```
public class Pessoa {  
    private String cpf;  
  
    public Pessoa(String cpfDaPessoa) {  
        cpf = cpfDaPessoa;  
    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa1 = new Pessoa("111.111.111-11");  
        Pessoa pessoa2 = pessoa1;  
        Pessoa pessoa3 = new Pessoa("111.111.111-11");  
  
        System.out.println(pessoa1 == pessoa2);  
        System.out.println(pessoa1.equals(pessoa2));  
  
        System.out.println(pessoa1 == pessoa3);  
        System.out.println(pessoa1.equals(pessoa3));  
    }  
}
```

Console Problems Javadoc Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\Java\j

```
true  
true  
false  
false
```

Saída no console

Sobrescrevendo o equals()

```
public class Pessoa {  
    private String cpf;  
  
    public Pessoa(String cpfDaPessoa) {  
        cpf = cpfDaPessoa;  
    }  
  
    public boolean equals(Pessoa pessoa) {  
        return cpf.equals(pessoa.cpf);  
    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa1 = new Pessoa("111.111.111-11");  
        Pessoa pessoa2 = pessoa1;  
        Pessoa pessoa3 = new Pessoa("111.111.111-11");  
  
        System.out.println(pessoa1 == pessoa2);  
        System.out.println(pessoa1.equals(pessoa2));  
  
        System.out.println(pessoa1 == pessoa3);  
        System.out.println(pessoa1.equals(pessoa3));  
    }  
}
```

Método equals ()
sobrescrito

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\Java\jdk-8.0.60\bin\java.exe

true
true
false
true

Nova saída no
console

Sobrecarga e Sobrescrita

- Sobrecarga e sobrescrita de métodos são conceitos fundamentais na orientação a objetos que serão mais bem explorados quando estudarmos **Herança** e na disciplina **Padrões de Projeto**

A morte de objetos

- Conversamos até agora sobre a criação de objetos, mas como os objetos morrem? Quando eles deixam de existir? O que acontece quando a memória está cheia de objetos inúteis?
- O gerenciamento de memória é essencial em muitos aplicativos
 - Memória é um recurso finito

Coletor de Lixo

- O propósito da coleta de lixo é descartar os objetos que não podem mais ser acessados.
 - Um objeto se torna elegível para a Coleta de Lixo quando não há mais referência a ele.
 - Não há garantias sobre quando a JVM vai acionar o Coletor de Lixo
- É possível tentar forçar a execução do Coletor de Lixo com o método **System.gc()**;
- Como tornar um objeto elegível para a coleta de lixo?
 - Resposta: Tornando ele inacessível!

Coletor de Lixo

Criação de um novo objeto do tipo Pessoa no heap.

```
public static void main(String[] args) {  
    Pessoa pessoa = new Pessoa();  
    pessoa = null;  
}
```

Agora, a variável de referência que apontava para esse objeto tem seu valor trocado para **null**. O objeto criado anteriormente torna-se inacessível e, portanto, elegível para coleta de lixo.

Métodos

- O que os métodos representam?
- Os métodos podem ou não ter retorno
- Os métodos podem ou não ter parâmetros de entrada

Sintaxe para Declaração de Métodos

Modificador
de Acesso

Tipo de
Retorno

Nome do
Método

}

}

Exemplo de Métodos para a Classe Professor

```
//método sem retorno  
public void prepararAula() {
```

```
}
```

```
//método com retorno  
public int darNota() {  
    return 10;
```

```
}
```

Exemplo de Métodos para a Classe Professor

//método com um parâmetro de entrada

```
public String perguntar(String assunto) {  
    return "O que é " + assunto + " ?";  
}
```

//método com mais de um parâmetro de entrada

```
public void alunoAprovado(String nome, int nota1, int nota2) {  
    float media = (float) (nota1+nota2)/2;  
    if (media >= 7)  
        System.out.println(nome + " foi aprovado");  
    else  
        System.out.println(nome + " foi reprovado");  
}
```

Nomes de Métodos válidos

1. Não pode ser uma palavra reservada
2. Não pode conter espaços em branco
3. Os nomes de **Métodos** devem começar com
 1. uma letra
 2. um cifrão \$
 3. um underline _
4. O restante do nome pode ser composto por letras, números, cifrão ou underline
5. Java é *case sensitive*

Convenções para Nomenclatura de Métodos

- Os nomes de **Métodos** geralmente são verbos
- O nome do método devem ser curtos mais significativos
- A nomenclatura dos Métodos segue o padrão *Camel Case*, com a primeira palavra ficando toda em minúsculo
- No caso do método 'get' para um atributo do tipo booleano, pode ser utilizado 'is'

Exercício

- Caracterize os nomes dos **métodos** como (1) válido ou não válido e (2) seguindo ou não as convenções de nomenclatura de classes que aprendemos

- | | | |
|------------------|-------------------|------------------------|
| a) getCliente | i) fabricarObjeto | q) _setProfessor_ |
| b) is_ativo | j) setR\$ | r) next_line |
| c) isAtivo | k) salvar | s) isFim |
| d) \$conectarBD | l) lerArquivoTXT | t) getDataDeNascimento |
| e) desconectar | m) setCEP | u) setAnodeNascimento |
| f) recuperarNOME | n) isFalse | v) voarVoarSubirSubir |
| g) Deletar | o) PLAY | w) getProgramação |
| h) cachorro | p) starApp | x) pause |

Passagem de Parâmetros

- Em programação, no geral, há duas formas de passar parâmetros para funções
 - Por valor
 - Por referência

Parâmetros por Valor

- Quando se passa um valor de um atributo para um método, uma cópia do valor que está neste atributo é passada.
- Desta forma, qualquer alteração no valor dessa variável, realizada dentro do método , será feita em uma cópia, que não tem ligação direta com o atributo passado por parâmetro.
- **PASSAGEM DE PARÂMETRO EM JAVA É SEMPRE POR VALOR**

Parâmetros por Referência

- **EM JAVA, PASSAGEM DE PARÂMETRO É SEMPRE POR VALOR**
- Entretanto, ao passar para uma função uma cópia de uma referência ao objeto, fornece-se um acesso direto a aquele objeto.
- Assim, ao passar uma variável de referência, dependendo das operações de edição que serão realizadas, pode-se alterar o estado do objeto

Exemplo de Passagem de Parâmetros

```
public class Pessoa {  
    private int idade;  
    private String nome;  
  
    public Pessoa(int idade, String nome) {  
        this.idade = idade;  
        this.nome = nome;  
    }  
  
    public void fazerAniversario(int idadeAtual) {  
        idadeAtual++;  
    }  
  
    public void fazerAniversario(Pessoa pessoa) {  
        pessoa.idade++;  
    }  
  
    public int getIdade() {  
        return idade;  
    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa(25, "cleyton");  
  
        pessoa.fazerAniversario(pessoa.getIdade());  
        System.out.println("Sua idade é " + pessoa.getIdade());  
  
        pessoa.fazerAniversario(pessoa);  
        System.out.println("Sua idade é " + pessoa.getIdade());  
    }  
}
```

Métodos Sobrecarregados: o 1º recebe um parâmetro inteiro e o incrementa em uma unidade; o 2º recebe um objeto e incrementa o valor sua propriedade idade em uma unidade.

Chamada aos Métodos Sobrecarregados: na 1ª chamada, passa-se uma variável primitiva; na 2ª chamada, passa-se uma variável de referência

A primeira chamada ao método fazerAniversario, não altera o valor da propriedade idade no objeto, pois é passada uma cópia do valor. A segunda altera, pois é passada uma cópia do valor usado para acessar ao objeto, mas que serve para acessar diretamente ao objeto.

Console Problems Javadoc Declaration

<terminated> Pessoa (3) [Java Application] C:\Program Files (x86)\J

Sua idade é 25
Sua idade é 26

Atributos

- O que os atributos representam?
- Convenção
 - Atributos privados
 - Métodos para acessar os atributos privados
 - Getters: recupera o valor de um atributo
 - Setters: configuram o valor de um atributo

Sintaxe para Declaração de Atributos de uma Classe

Modificador
de Acesso

Tipo do
Atributo

Nome do
Atributo

;

Exemplos de Atributos para a Classe Professor

```
private String nome;  
private String formacao;  
private float salario;  
private int idade;  
private String celular;
```

Nomes de Atributos válidos

1. Não pode ser uma palavra reservada
2. Não pode conter espaços em branco
3. Os nomes de **Atributos** devem começar com
 1. uma letra
 2. um cifrão \$
 3. um underline _
4. O restante do nome pode ser composto por letras, números, cifrão ou underline
5. Java é *case sensitive*

Convenções para Nomenclatura de Atributos

- O nome de **Atributos** e **Variáveis** também devem seguir o padrão *Camel Case*
 - Recomenda-se que os nomes sejam curtos e significativos
- No caso de **Constantes**, utiliza-se apenas caixa alta.
 - Para as **Constantes** com múltiplas palavras em seu nome, utiliza-se underline para separar as palavras.

Exercício

- Caracterize os nomes de **atributo** como (1) válido ou não válido e (2) seguindo ou não as convenções de nomenclatura de classes que aprendemos

- | | | |
|-------------------|----------------|-----------------------|
| a) cliente | i) New | q) _Professor_ |
| b) semestreLetivo | j) float | r) BigLong |
| c) pe\$o | k) Inteiro | s) Blob |
| d) \$4L4r10 | l) CPF | t) dataDeNascimento |
| e) \$_1 | m) CEP | u) \$GRAVIDADE |
| f) contador | n) true | v) string |
| g) nome | o) Conta | w) isso_não_é_legal |
| h) MelhoresAmigos | p) Programação | x) CONSTANTE_AVOGRADO |

Tipos de Variáveis

- Um atributo ou variável em Java pode ser de um desses dois tipos
 - Tipo primitivo
 - Tipo de referência

Tipos Primitivos e Tipos de Referência

- Tipos Primitivos
 - boolean, char, float, double, byte, int, long
- Tipos de Referência
 - do tipo de alguma classe

Tipos Primitivos e de Referência

```
public class Aluno {  
  
    private int idade;  
    private float CRE;  
  
    private Curso curso;  
  
    private String nome;  
    private String cpf;  
  
}
```

Escopo de Variáveis

- O que é escopo?
 - É o nome que se dá aos “limites” de uma variável
 - Trata dos locais onde a variável “existe” (i.e., dos locais onde uma variável é “visível”)
- O escopo de uma variável é a região do programa onde cada variável pode ser referenciada pelo seu nome.
- O escopo também determina quando o sistema aloca e libera o espaço de memória para a variável.
- O escopo é diferente do conceito de visibilidade de OO
 - Visibilidade se aplica apenas a atributos de uma classe
 - Visibilidade é definida utilizando modificadores de acesso

Escopo de Variáveis

- Quais escopos existem em Java?
 - Escopo de Instância
 - Escopo de Método
 - Escopo de Bloco
 - Escopo de Classe

Escopo de Instância

- Uma variável que tem escopo de classe pode ser utilizada por todos os métodos da classe
- Ela “existe” em todo o corpo da classe
- **Os atributos de uma classe tem escopo de instância**

Escopo da variável 'nome'

```
public class ExemploEscopoInstancia {  
    private String nome;  
  
    public void metodo1() {  
    }  
  
    public void metodo2() {  
    }  
}
```

A palavra reservada `this`

- A palavra reservada `this` permite que dentro da classe se referencie explicitamente os próprios recursos da classe (atributos e métodos) de forma mais legível e, frequentemente, menos ambígua.
- A palavra reservada `this` só funciona para variáveis com escopo de instância (i.e., atributos), mas também pode ser utilizada com métodos da própria classe

A palavra reservada **this**

- Pode-se utilizar o **this** para referenciar propriedades

```
public void setPeso(float novoPeso) {  
    this.peso = novoPeso;  
}
```

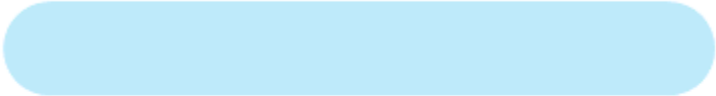
- Ou mesmo métodos

```
public int fazerConta(int numero1, int numero2, String op) {  
    switch (op) {  
        case "+": return this.somar(numero1, numero2);  
        case "-": return this.subtrair(numero1, numero2);  
        case "*": return this.multiplicar(numero1, numero2);  
        case "/": return this.dividir(numero1, numero2);  
    }  
    return 0;  
}
```

Escopo de Método

- Variáveis com escopo de método só podem ser “enxergadas” dentro do seu método
 - Também chamado de Escopo Local
- **Parâmetros de entrada e variáveis declaradas dentro de métodos tem escopo local**
 - No caso de variáveis declaradas dentro do método, seu escopo se restringe do ponto de sua declaração até o fim do método

Escopo da variável 'x'

```
public class ExemploEscopoMetodo {  
  
    public void metodo1(int x) {  
          
    }  
  
    public void metodo2() {  
        int y;  
    }  
}
```

Escopo de Método

- Qual a área em que a variável x é visível?

```
public void metodo3(int x) {  
    if (x%2 == 0) {  
        System.out.println("x é par");  
    }  
  
    int y = -1;  
  
    x-=y;  
  
    System.out.println("x = "+ y);  
}
```


Escopo de Método

- Qual a área em que a variável x é visível?

```
public void metodo3(int x) {  
    if (x%2 == 0) {  
        System.out.println("x é par");  
    }  
  
    int y = -1;  
  
    x-=y;  
  
    System.out.println("x = "+ y);  
}
```

Escopo de Método

- Qual a área em que a variável `y` é visível?

```
public void metodo3(int x) {  
    if (x%2 == 0) {  
        System.out.println("x é par");  
    }  
  
    int y = -1;  
  
    x-=y;  
  
    System.out.println("x = "+ y);  
}
```

Escopo de Método

- Qual a área em que a variável `y` é visível?

```
public void metodo3(int x) {  
    if (x%2 == 0) {  
        System.out.println("x é par");  
    }  
  
    int y = -1;  
  
    x-=y;  
  
    System.out.println("x = "+ y);  
}
```

Escopo de Bloco

- Em Java, é permitido declarar novas variáveis dentro de blocos de início e fim '{ }'
 - Variáveis declaradas dentro de blocos, só existem dentro do bloco de início e fim que foram declaradas

Escopo de Bloco

- A variável **y** só é visível dentro do bloco do if
- A variável **i** só é visível dentro do bloco do for
- A variável **z** só é visível dentro do bloco do switch case

```
public class ExemploEscopoBloco {  
  
    public void metodo1(int x) {  
  
        if (x == 1) {  
            int y = x;  
            y++;  
        }  
  
        for (int i=0; i<10;i++)  
            System.out.println(i);  
  
        switch (x) {  
            case 0: int z = 1;  
                break;  
            case 1: z = 2;  
                break;  
            default: z=3;  
                break;  
        }  
    }  
}
```

A palavra reservada **this** – parte 2

- Também é possível utilizar o **this** para desambiguar o código.
 - O que significa desambiguar?
- É possível criar variáveis com o mesmo nome dos atributos da classe dentro do Escopo de Método e do Escopo de Bloco .
 - Nesse caso, para referir-se ao atributo da instância, utiliza-se a palavra reservada **this**.

A palavra reservada **this** – parte 2

Aqui, está acontecendo o seguinte: o atributo nome do objeto está recebendo o valor da variável nome passada como parâmetro

Aqui, estou criando uma variável com o escopo de método com o mesmo nome de um dos atributos da classe Pessoa.

Para comparar o valor do atributo 'nome', que tem escopo de instância, com a variável 'nome', que tem escopo de método, é obrigatório usar o **this** para referenciar o atributo do objeto.

```
public class Pessoa {  
    private String nome;  
  
    public void setName(String nome) {  
        this.nome = nome;  
    }  
  
    public boolean temOMesmoNome(Pessoa outra) {  
        String nome = outra.nome;  
  
        //se as duas pessoas tem o mesmo nome  
        if (this.nome.equals(nome))  
            return true; //retorne true  
        else //senao  
            return false; // retorne false  
  
        //outra possibilidade  
        //return this.nome.equals(nome);  
    }  
}
```

O parâmetro de entrada 'nome' tem o mesmo nome do atributo da Classe pessoa.

Escopo de Classe

- Algumas vezes, temos a necessidade de que todas as classes compartilhem de uma mesma variável.
- Uma variável de classe é uma variável cujo valor é comum a todos os objetos representantes da classe. Mudar o valor de uma variável de classe em um objeto automaticamente muda o valor para todos os objetos instâncias da mesma classe.
 - Um exemplo óbvio de uma variável de classe seria o número de instâncias desta classe que já foram criadas.
- Variáveis com Escopo de Classe também são chamadas de Variáveis Estáticas
 - Variáveis com Escopo de Classe “pertencem” à classe

Variáveis Estáticas

```
public class Pessoa {  
    public static int numeroDePessoas;  
  
    public Pessoa() {  
        Pessoa.numeroDePessoas++;  
    }  
  
    public static void main(String[] args) {  
        System.out.println("Existem no mundo " + Pessoa.numeroDePessoas);  
  
        Pessoa pessoa = new Pessoa();  
  
        System.out.println("Existem no mundo " + Pessoa.numeroDePessoas);  
    }  
}
```

Para declarar uma variável estática, basta utilizar a palavra reservada static

Para utilizar a variável estática, deve-se utilizar o nome da classe, ponto, seguido pelo nome da variável

Console Problems @ Javadoc Declar

```
<terminated> Pessoa (3) [Java Application] C:\Program F  
Existem no mundo 0  
Existem no mundo 1
```

A palavra reservada **static**

- A palavra **static** aplicada em um atributo significa que esse atributo pertence à Classe, i.e., todas as instâncias de objetos dessa classe vão compartilhar um mesmo atributo
 - Esse atributo estático também é chamado de variável estática
- Para acessar uma **atributo estático** utiliza-se o nome da classe, ponto, o nome do atributo.
- **Que atributo estáticos vocês veem usando frequentemente?**

A palavra reservada **static**

- Que atributos estáticos vocês veem usando frequentemente?

Existe uma classe chamada System nas bibliotecas padrão de Java

Mas, o que a gente sabe sobre o objeto 'out' é que ele tem um método chamado println que serve para imprimir mensagens no console.

```
System.out.println("Olá mundo!");
```

Essa classe System possui uma variável estática chamada 'out', que é do tipo de uma classe que a gente não conhece.

A palavra reservada **static**

- A palavra **static** também pode ser usado para criar **métodos estáticos**
 - O que são métodos estáticos?
 - Métodos que são acessados sem a necessidade de uma instância da classe.
- Qual a utilidade dos métodos estáticos?

A palavra reservada **static**

- Imagine que você tem um método que sempre faz a mesma coisa e que não depende do estado do objeto (i.e., dos valores dos atributos)
 - Gerar um número aleatório
 - Dizer a hora exata
 - Informar o dia da semana por extenso
 - Exibir uma mensagem formatada na tela
- **Métodos estáticos não podem utilizar variáveis com escopo de instância!**

Métodos Estáticos

Isso é um método estático!

Ele se tornou um método estático, pois foi utilizada a palavra reservada 'static' para modificá-la.

```
public class Pessoa {
```

```
    public static String dizerOi() {  
        return "oi";  
    }
```

Para invocar um método estático, usa-se o nome da classe, ponto, o nome do método estático

```
    public static void main(String[] args) {  
        System.out.println(Pessoa.dizerOi());  
    }  
}
```

Constantes em Java

- O que são constantes?
 - Variáveis que não mudam de valor
- Para definir uma variável como constante (i.e., cujo valor não muda) basta usar a palavra reservada **final**
- **Constantes devem ser Variáveis Estáticas**

```
public class Quadrilatero {  
    public static final int NUMERO_DE_LADOS = 4;  
}
```

Dissecando uma classe

	<pre>public class Pessoa {</pre>	
	<pre> private String nome;</pre>	Atributos
Construtores	<pre> public Pessoa() { } public Pessoa(String nome) { this.nome = nome; }</pre>	
Métodos da classe Pessoa	<pre> public String getNome() { return nome; } public void setNome(String nome) { this.nome = nome; }</pre>	Métodos getters e setters
	<pre> public String ola(String nome) { return "Olá " + nome + " meu nome é " + this.nome; } }</pre>	

Modificadores de Acesso

- Restringem o acesso a um **atributo** ou **método** de uma classe ou à própria **classe**
- Os modificadores de acesso podem controlar o acesso à
 - **Classes**
 - **Atributos**
 - **Métodos**

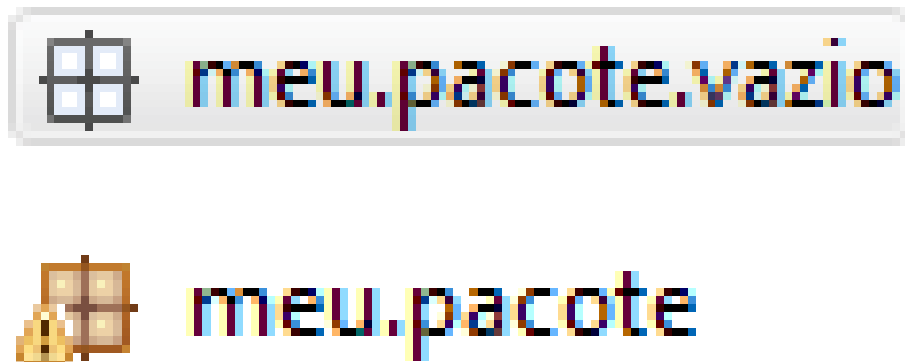
Modificadores de Acesso

Modificador de Acesso	Quem tem acesso?
public	A classe, método ou atributo pode ser enxergado por qualquer um.
private	Esse modificador não se aplica às classes. Somente para métodos e atributos. Os membros da classe definidos como private não podem ser visualizados fora da classe.
protected	Esse modificador não se aplica às classes. Somente para métodos e atributos. O modificador protected torna o método ou atributo acessível às classes do mesmo <u>PACOTE</u> ou através de herança.
nenhum/default/friendly	A classe, método ou atributo é acessível somente por classes do mesmo <u>PACOTE</u> , na sua declaração não é definido nenhum tipo de modificador, sendo este aplicado pelo compilador.

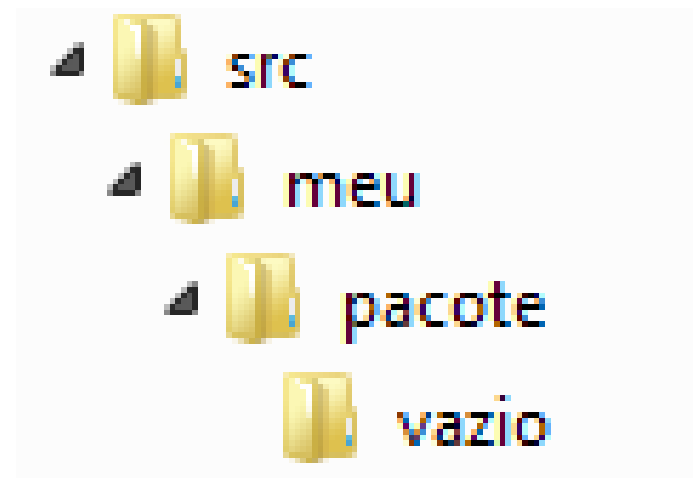
Pacotes em Java

- Um pacote funciona como uma pasta para organizar Classes
- No Sistema Operacional os pacotes funcionam exatamente como pastas para o código fonte das classes

Como é no Eclipse



Como é no Sistema de Pastas



Pacotes em Java

- Uma das convenções dos programadores é utilizar os pacotes para manter as Classes organizadas
 - Não há uma convenção para a nomenclatura dos pacotes, mas aconselha-se usar nomes que seja descritivos
- De acordo com os modificadores de acesso utilizados, alguns recursos podem não ser visíveis fora do pacote.

Visibilidade - Demonstração

- Roteiro

- Crie um novo projeto no Eclipse

- Crie dois pacotes

-  pacote1

-  pacote2

- Crie três classes, duas no pacote1 e uma no pacote2, todas públicas

-  pacote1

-  Classe1.java

-  Classe2.java

-  pacote2

-  Classe3.java

Visibilidade - Demonstração

- Crie um método algo() e um main da seguinte forma

```
public class Classe1 {  
    public void algo() {  
    }  
  
    public static void main(String[] args) {  
        Classe1 classe1 = new Classe1();  
        classe1.algo();  
        Classe2 classe2 = new Classe2();  
        classe2.algo();  
        Classe3 classe3 = new Classe3();  
        classe3.algo();  
    }  
}
```

Visibilidade - Demonstração

- Altere os modificadores de acesso dos métodos e classes e observe como isso afeta o projeto.
 - Exemplos
 - Mude a visibilidade do método algo() da Classe3 para **protected**
 - Mude a visibilidade da Classe3 para a **default**
 - Mude a visibilidade da Classe1 para a **default**
 - Mude a visibilidade do método algo() da Classe1 para **default**

Próximas Aulas

- Algumas bibliotecas em Java
 - String
 - Datas
 - Envelopadores
 - Math
 - Coleções: ArrayList, HashSet, HashMap