

Construtores e Especificadores de Acesso

Professora:
Fátima L. S. Nunes



Orientação a Objetos

- Conceitos do Paradigma de Orientação a Objetos já vistos:
 - Classes – atributos e métodos
 - Objetos – instanciação de classes
 - Programas com vários objetos
 - Variáveis e Memória
- Novos conceitos:
 - Construtores
 - Especificadores de acesso

Construtores



Construtores

- O que é um objeto?

Construtores

- O que é um objeto?
 - Instanciação de uma classe

Construtores

- O que é um objeto?
 - Instanciação de uma classe
- Diferentes tipos de objetos...
 - Para criá-los: ???

Construtores

- O que é um objeto?
 - Instanciação de uma classe
- Diferentes tipos de objetos...
 - Para criá-los:

ClasseDefinida cd = new ClasseDefinida()

Variável cd ⇒ referência a um novo objeto da ClasseDefinida

- A classe pode não ter atributos (ex: ConversorDeTemperaturas)
 - Quando é interessante ter atributos?
- Se a classe tem atributos, escrevemos métodos obtém/altera para cada atributo
- Portanto, após criarmos um objeto, precisamos inicializar seus atributos

- A classe pode não ter atributos (ex: ConversorDeTemperaturas)
 - Quando é interessante ter atributos?
 - Quando queremos armazenar valores para uso futuro
- Se a classe tem atributos, escrevemos métodos obtém/altera para cada atributo
- Portanto, após criarmos um objeto, precisamos inicializar seus atributos

```
class ClasseDefinida
{
    tipo1 _atributo1;
    ...
    tipoN _atributoN;

    void alteraAtributo1 (tipo1 valor1)
    { _atributo1 = valor1; }

    tipo1 obtemAtributo1 () { return _atributo1; }

    ...

    void alteraAtributoN (tipoN valorN)
    { _atributoN = valorN; }

    tipoN obtemAtributoN() { return _atributoN; }
}
```

```
class ClasseDefinida  
{
```

```
    tipo1 _atributo1;
```

```
    ...
```

```
    tipoN _atributoN;
```

```
void alteraAtributo1 (tipo1 valor1)  
{ _atributo1 = valor1; }
```

```
tipo1 obtemAtributo1 () { return _atributo1; }
```

```
    ...
```

```
void alteraAtributoN (tipoN valorN)  
{ _atributoN = valorN; }
```

```
tipoN obtemAtributoN() { return _atributoN; }
```

```
}
```

```
ClasseDefinida cd = new ClasseDefinida();  
cd.alteraAtributo1(v1);  
...  
cd.alteraAtributoN(vn);
```

```
class ClasseDefinida  
{
```

```
    tipo1 _atributo1;
```

```
    ...
```

```
    tipoN _atributoN;
```

```
void alteraAtributo1 (tipo1 valor1)  
{ _atributo1 = valor1; }
```

```
tipo1 obtemAtributo1 () { return _atributo1; }
```

```
...
```

```
void alteraAtributoN (tipoN valorN)  
{ _atributoN = valorN; }
```

```
tipoN obtemAtributoN() { return _atributoN; }
```

```
}
```

```
ClasseDefinida cd = new ClasseDefinida();  
cd.alteraAtributo1(v1);  
...  
cd.alteraAtributoN(vn);
```

Não precisa ser sempre assim!
Podemos usar
CONSTRUTORES!

Construtores

- Em Java:
 - quando não especificamos como um objeto será criado, a linguagem fornece um **construtor padrão**.

Construtores

- Em Java:
 - quando não especificamos como um objeto será criado, a linguagem fornece um **construtor padrão**.
 - Inicializa atributos com...

Construtores

- Em Java:
 - quando não especificamos como um objeto será criado, a linguagem fornece um **construtor padrão**.
 - Inicializa atributos com... **valores default de cada tipo**

Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto p = new Produto();
p.imprimeDados();
```


Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto p = new Produto();
p.imprimeDados();
```

```
Preço = 0
Código = 0
```

Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto p = new Produto();
p.imprimeDados();
```

```
Preço = 0
Código = 0
```

Como definir valores prévios para variáveis? Ou como definir valores durante a *instanciação* objeto?

Construtores

Como definir valores prévios para variáveis? Ou como definir valores durante a *instanciação* objeto?

Conceito de construtor

```
class Produto
{
    double _preco;
    int _codigo;

    Produto (double preco, int cod)
    {
        _preco = preco;
        _codigo = cod;
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    Produto (double preco, int cod)
    {
        _preco = preco;
        _codigo = cod;
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto prod = new Produto(78.3,50);
prod.imprimeDados();
```

Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    Produto (double preco, int cod)
    {
        _preco = preco;
        _codigo = cod;
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto prod = new Produto(78.3,50);
prod.imprimeDados();
```

```
Preço = 78.3
Código = 50
```

Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    Produto (double preco, int cod)
    {
        _preco = preco;
        _codigo = cod;
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto prod = new Produto(78.3,50);
prod.imprimeDados();
```

Preço = 78.3
Código = 50

O que acontece se fizermos:

```
Produto prod = new Produto();
```

???????

Construtores

```
class Produto
{
    double _preco;
    int _codigo;

    Produto (double preco, int cod)
    {
        _preco = preco;
        _codigo = cod;
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
    }
}
```

Instanciação:

```
Produto prod = new Produto(78.3,50);
prod.imprimeDados();
```

Preço = 78.3
Código = 50

O que acontece se fizermos:

```
Produto prod = new Produto();
```

Static Error: No constructor in Produto accepts arguments ()

Construtores

- Algumas observações:
 - Quando é definido um construtor, **não é mais possível usar o construtor padrão** (não é possível criar objeto sem obedecer as regras do construtor que a classe disponibiliza)...
 - ... a não ser que você implemente um equivalente
 - Podemos ter vários construtores em uma classe

Mais de um Construtor na mesma classe

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;
```

```
    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }
```

```
    Produto (int cod, int qtd)
    {
        _codigo =cod;
        _quantidade = qtd;
    }
```

```
    Produto (int cod)
    {
        this (10,cod,100);
    }
```

```
    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
        System.out.println("Quantidade = " + _quantidade);
    }
```



Instanciações possíveis:

```
> Produto prod1 = new Produto(43.20, 54, 200);
> prod1.imprimeDados();
```

Mais de um Construtor na mesma classe

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;

    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }

    Produto (int cod, int qtd)
    {
        _codigo =cod;
        _quantidade = qtd;
    }

    Produto (int cod)
    {
        this (10,cod,100);
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
        System.out.println("Quantidade = " + _quantidade);
    }
}
```

Instanciações possíveis:

```
> Produto prod1 = new Produto(43.20, 54, 200);
> prod1.imprimeDados();
```

Preço = 43.2

Código = 54

Quantidade = 200



Mais de um Construtor na mesma classe

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;

    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }

    Produto (int cod, int qtd)
    {
        _codigo = cod;
        _quantidade = qtd;
    }

    Produto (int cod)
    {
        this (10,cod,100);
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
        System.out.println("Quantidade = " + _quantidade);
    }
}
```

Instanciações possíveis:

```
> Produto prod1 = new Produto(43.20, 54, 200);
> prod1.imprimeDados();
```

Preço = 43.2

Código = 54

Quantidade = 200

```
> Produto prod2 = new Produto(300,195);
> prod2.imprimeDados();
```

Preço = 0.0

Código = 300

Quantidade = 195



Palavra reservada this

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;

    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }

    Produto (int cod, int qtd)
    {
        _codigo =cod;
        _quantidade = qtd;
    }

    Produto (int cod)
    {
        this (10,cod,100);
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
        System.out.println("Quantidade = " + _quantidade);
    }
}
```



Palavra reservada this

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;

    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }

    Produto (int cod, int qtd)
    {
        _codigo =cod;
        _quantidade = qtd;
    }

    Produto (int cod)
    {
        this (10,cod,100);
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
        System.out.println("Quantidade = " + _quantidade);
    }
}
```

- É somente uma referência ao próprio objeto.
- Ou seja, se tivermos a chamada
Produto p = new Produto(50);
- O construtor ...

```
Produto (int c)
{
    this (10,c,100);
}
```
- ... repassa o trabalho para o outro construtor da mesma classe.

```
Produto (double preco, int cod, int qtd)
{
    _preco = preco;
    _codigo = cod;
    _quantidade = qtd;
}
```

Construtor com this

```
class Produto
```

```
{  
    double _preco;  
    int _codigo;  
    int _quantidade;
```

```
    Produto (double preco, int cod, int qtd)
```

```
{  
    _preco = preco;  
    _codigo = cod;  
    _quantidade = qtd;  
}
```

```
    Produto (int cod, int qtd)
```

```
{  
    _codigo =cod;  
    _quantidade = qtd;  
}
```

```
    Produto (int cod)
```

```
{  
    this (10,cod,100);  
}
```

```
    void imprimeDados()
```

```
{  
    System.out.println("Preço = " + _preco);  
    System.out.println("Código = " + _codigo);  
    System.out.println("Quantidade = " + _quantidade);  
}
```

```
> Produto prod3 = new Produto(400);  
> prod3.imprimeDados();
```



Construtor com this

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;

    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }

    Produto (int cod, int qtd)
    {
        _codigo =cod;
        _quantidade = qtd;
    }

    Produto (int cod)
    {
        this (10,cod,100);
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " + _codigo);
        System.out.println("Quantidade = " + _quantidade);
    }
}
```

```
> Produto prod3 = new Produto(400);
> prod3.imprimeDados();
Preço = 10.0
Código = 400
Quantidade = 100
```



Palavra reservada this – outro uso

- Parâmetro com mesmo nome que atributo do objeto

```
class Produto
{
    double preco;
    int codigo;
    int quantidade;

    Produto (int cod, double pr, int quantidade )
    {
        preco = pr;
        codigo = cod;
        this.quantidade = quantidade;
    }
}
```


Palavra reservada this – outro uso

- Parâmetro com mesmo nome que atributo do objeto

```
class Produto
{
    double preco;
    int codigo;
    int quantidade;
```

```
    Produto (int cod, double pr, int quantidade )
    {
        preco = pr;
        codigo = cod;
        this.quantidade = quantidade;
    }
}
```

Deixa claro que se trata de um atributo do objeto, pois o parâmetro recebido tem o mesmo nome. Precisa diferenciar.

Construtores

- Quando há mais de um construtor na classe:
 - Têm o mesmo nome – diferem apenas nos parâmetros.
 - Quando invocado, o **compilador Java escolhe o construtor correto** conforme a assinatura (tipos dos parâmetros e sua ordem).
 - Esta técnica também pode ser usada para métodos com mesmo nome e que recebem parâmetros diferentes: *polimorfismo de nome*.

Exercício

- Crie uma classe Aluno com os atributos: numeroUSP, ano de matrícula (inteiros) e média ponderada (double). Implemente os seguintes métodos para esta classe:
 - imprimeAluno: imprime atributos
 - construtores:
 - a) recebe número e ano de matrícula e atualiza atributos correspondentes;
 - b) recebe e atualiza somente média ponderada;
 - c) recebe somente número (neste caso, deve assumir que o ano de matrícula é o ano corrente e chamar o construtor correspondente).

Especificadores de acesso

Especificadores de acesso

- Para exemplificar os especificadores de acesso, vamos abordar um problema de segurança...
- Suponhamos que um usuário crie um objeto de Produto.
- O que acontece após a execução deste código?

```
Produto prod = new Produto(23.4, 45, 900)
prod.imprimeDados();
prod._quantidade = prod._quantidade - 100;
prod.imprimeDados();
```

```
class Produto
{
    double _preco;
    int _codigo;
    int _quantidade;

    Produto (double preco, int cod, int qtd)
    {
        _preco = preco;
        _codigo = cod;
        _quantidade = qtd;
    }

    void imprimeDados()
    {
        System.out.println("Preço = " + _preco);
        System.out.println("Código = " +
        _codigo);
        System.out.println("Quantidade = " +
        _quantidade);
    }
}
```

Especificadores de acesso

```
Produto prod = new Produto(23.4, 45, 900)  
prod.imprimeDados();  
prod._quantidade = prod._quantidade - 100;  
prod.imprimeDados();
```

- Temos um problema de segurança!
- Não podemos permitir que qualquer usuário altere a quantidade de mercadorias em estoque !!! (E se não tiver produto suficiente???? → sistema em estado inconsistente)
- Como resolver ???

Especificadores de acesso

- Para evitar problemas como este, podemos usar proteções explícitas, indicando que somente métodos do próprio objeto alterem os seus atributos.

Especificadores de acesso

- Então, vamos incluir os métodos *subtraiEstoque* e *adicionaEstoque* na classe Produto, responsáveis por alterar a quantidade existente do produto nos estoques da empresa.
- Como seria o método *subtraiEstoque* ???

Especificadores de acesso

```
void subtraiEstoque(int qtde)
{
    if (qtde < 0)
    {
        System.out.println("Não é permitido quantidade negativa.");
        return;
    }
    if (qtde > _quantidade)
    {
        System.out.println("Quantidade em estoque insuficiente.");
        imprimeDados();
    }
    else
    {
        _quantidade = _quantidade - qtde;
        System.out.println("Atualização realizada");
    }
}
```

Especificadores de acesso

- E o método adicionaEstoque ???

Especificadores de acesso

- E o método adicionaEstoque ???

```
void adicionaEstoque(int qtde)
{
    if (qtde < 0)
        System.out.println("Não é permitido quantidade negativa.");
    else
    {
        _quantidade = _quantidade + qtde;
        System.out.println("Atualização realizada.");
    }
}
```

Especificadores de acesso - private

- Com os métodos construídos, usamos a proteção explícita, definindo o atributo com palavra chave **private**

```
class Produto
{
    private double _preco;
    private int _codigo;
    private int _quantidade;

    ....
}
```

Especificadores de acesso - private

- Com os métodos construídos, usamos a proteção explícita, definindo o atributo com palavra chave **private**

```
class Produto
{
    private double _preco;
    private int _codigo;
    private int _quantidade;

    ....
}
```

Somente métodos pertencentes à mesma classe poderão alterar os atributos.

Especificadores de acesso - private

```
class Produto  
{  
    private double _preco;  
    private int _codigo;  
    private int _quantidade;
```

....

```
void subtraiEstoque(int qtde) {  
    if (qtde > _quantidade) {  
        System.out.println("Quantidade em estoque insuficiente.");  
        imprimeDados();  
    }  
    else {  
        _quantidade = _quantidade - qtde;  
        System.out.println("Atualização realizada");  
    }  
}
```

Método subtraiEstoque pertence à mesma classe. Pode alterar o atributo *private*.

```
void adicionaEstoque(int qtde)  
{  
    if (qtde < 0)  
        System.out.println("Não é permitido quantidade negativa.");  
    else {  
        _quantidade = _quantidade + qtde;  
        System.out.println("Atualização realizada.");  
    }  
}
```

Método adicionaEstoque pertence à mesma classe. Pode alterar o atributo *private*.



Especificadores de acesso - public

- Contrário de private = **public**
- A palavra chave **public** permite acesso irrestrito aos atributos.

```
class Produto
{
    public double _preco;
    public int _codigo;
    public int _quantidade;

    ....
}
```

Boa prática de programação

- Normalmente usamos
 - private em cada atributo
 - métodos public de acesso a esses atributos

- Ex:

```
class X
```

```
{
```

```
    private tipo _atributoY;
```

```
    public tipo obtemAtributoY() { return _atributoY; }
```

```
    public void alteraAtributoY(tipo atr) { _atributoY = atr; }
```

```
}
```


Especificadores de acesso - private

```
class Produto
```

```
{  
    private double _preco;  
    private int _codigo;  
    private int _quantidade;
```

```
....
```

```
public void subtraiEstoque(int qtde) {  
    if (qtde > _quantidade) {  
        System.out.println("Quantidade em estoque insuficiente.");  
        imprimeDados();  
    }  
    else {  
        _quantidade = _quantidade - qtde;  
        System.out.println("Atualização realizada");  
    }  
}
```

Métodos **subtraiEstoque** e **adicionaEstoque** agora são **públicos** — todos podem executá-los.

```
public void adicionaEstoque(int qtde)  
{  
    if (qtde < 0)  
        System.out.println("Não é permitido quantidade negativa.");  
    else {  
        _quantidade = _quantidade + qtde;  
        System.out.println("Atualização realizada.");  
    }  
}
```



Especificadores de acesso - private

```
class AlteracaoProduto
{
    void alteraProduto()
    {
        Produto prod = new Produto(67.9,34, 900);
        prod._quantidade = prod._quantidade - 800;
    }
}
```



O que acontece?

Especificadores de acesso - private

```
class AlteracaoProduto
{
    void alteraProduto()
    {
        Produto prod = new Produto(67.9,34, 900);
        prod._quantidade = prod._quantidade - 800;
    }
}
```

**ERRO DE
COMPILAÇÃO !!!!**

File: AlteraProduto.java [line: 6]
Error: AlteraProduto.java:6: _quantidade has private access in Produto

Especificadores de acesso - private

```
class AlteracaoProdutoCorreta
{
    void alteraProduto()
    {
        Produto prod = new Produto(67.9,34, 900);
        prod.subtraiEstoque(800);
        prod.imprimeDados();
        prod.adicionaEstoque(153);
        prod.imprimeDados();
    }
}
```

Instanciação:

```
> AlteracaoProdutoCorreta apc = new
AlteracaoProdutoCorreta();
```

```
> apc.alteraProduto()
```

Atualização realizada

Preço = 67.9

Código = 34

Quantidade = 100

Atualização realizada

Preço = 67.9

Código = 34

Quantidade = 253



Especificadores de acesso

- Os especificadores de acesso também podem ser usados com os métodos:
- Exemplo:
`private imprimeDados()` – somente métodos daquele objeto podem invocar este método
- Quando nenhum especificador é usado:
 - Java usa o acesso *friendly* = permite a visibilidade somente dentro do mesmo pacote (pacote \Leftrightarrow diretório)

Especificadores de acesso

- Exemplos no DrJava
- Classes: Produto, Especificadores

Exemplo de aplicabilidade

- Você escreveu uma classe Estoque
 - vetor de Produtos (public)
 - número de produtos cadastrados
 - Suponha que você confia plenamente que os seus usuários (da classe Estoque) vão manter consistentes o vetor e o número de produtos cadastrados (Hahá – *bad idea...*)
 - Eles sabem que podem inserir novos produtos sequencialmente, pois a busca será sequencial
 -

Exemplo de aplicabilidade

- Você escreveu uma classe Estoque
 - vetor de Produtos (public)
 - número de produtos cadastrados
 - Suponha que você confia plenamente que os seus usuários (da classe Estoque) vão manter o vetor e o nr de produtos cadastrados consistente (Hahá – *bad idea...*)
 - Eles sabem que podem inserir novos produtos sequencialmente, pois a busca será sequencial
 - Você percebe que, com o crescimento do estoque a busca está ficando lenta, e resolve usar busca binária → manter o vetor ordenado
 - Todos os programas que usam a classe Estoque devem ser analisados para alterar a estratégia de inserção dos produtos
 - Você deve confiar que os produtos sempre são inseridos de forma ordenada

Boa prática de programação

- **ENCAPSULAMENTO!!!**
- Exemplo:
 - Classe Estoque
 - Tem um conjunto de produtos
 - Vetor? Não preciso saber
 - Métodos fornecem uma **interface** para a aplicação (API)
 - Método para consultar um produto
 - Busca binária? Busca sequencial? Não importa!
 - Métodos para inserir e remover um produto
 - Mantém os produtos ordenados? Não sei!
 - Se o desenvolvedor quiser mudar a implementação não afeta os usuários da API (nem mesmo você!!!), pois estes não dependem da implementação interna

Especificadores de acesso - static

- Especificador **static** indica quais atributos devem ser considerados pertencentes à classe e não específicos a cada objeto

```
class TesteStatic
{
    static int _quantidade = 0; //atributo de classe
    TesteStatic()
    {
        System.out.println("Criando um objeto do tipo TesteStatic");
        _quantidade++;
        System.out.println("Quantidade de objetos do tipo TesteStatic criados até agora:" + _quantidade);
    }
}
```

Especificadores de acesso

- Exemplos no DrJava
- Classe: TesteStatic

```
class TesteStatic
{
    static int _quantidade = 0;
    TesteStatic()
    {
        System.out.println("Criando um
objeto do tipo TesteStatic");
        _quantidade++;
        System.out.println("Quantidade de
objetos do tipo TesteStatic criados até
agora:" + _quantidade);
    }
}
```

```
> TesteStatic ts = new TesteStatic()
Criando um objeto do tipo TesteStatic
Quantidade de objetos do tipo TesteStatic criados até agora:1
> TesteStatic ts = new TesteStatic()
Criando um objeto do tipo TesteStatic
Quantidade de objetos do tipo TesteStatic criados até agora:2
> TesteStatic ts = new TesteStatic()
Criando um objeto do tipo TesteStatic
Quantidade de objetos do tipo TesteStatic criados até agora:3
```

Especificadores de acesso

- Exemplos no DrJava
- Classe: TesteSemStatic

```
class TesteSemStatic
{
    int _quantidade = 0;
    TesteSemStatic()
    {
        System.out.println("Criando um
objeto do tipo TesteStatic");
        _quantidade++;
        System.out.println("Quantidade
de objetos do tipo TesteSemStatic
criados até agora:" + _quantidade);
    }
}
```

```
> TesteSemStatic tss= new TesteSemStatic()
Criando um objeto do tipo TesteStatic
Quantidade de objetos do tipo TesteSemStatic criados até
agora:1
> TesteSemStatic tss= new TesteSemStatic()
Criando um objeto do tipo TesteStatic
Quantidade de objetos do tipo TesteSemStatic criados até
agora:1
> TesteSemStatic tss= new TesteSemStatic()
Criando um objeto do tipo TesteStatic
Quantidade de objetos do tipo TesteSemStatic criados até
agora:1
```

Especificadores de acesso - static

- Métodos **static** também são métodos da classe
 - podem ser chamados mesmo se não houver objetos da classe criados
- Exemplo: métodos da classe *Math*
 - podemos executar *Math.sin(double x)*, mesmo sem ter instanciado a classe *Math*.

Especificadores de acesso

- Exemplos no DrJava
- Classe: ExemploMetodoStatic

```
class ExemploMetodoStatic
{
    static int _quantidade = 768;
    static void imprimeValor()
    {
        System.out.println("Valor do atributo
        _quantidade:" + _quantidade);
    }
}
```

> ExemploMetodoStatic.imprimeValor()
Valor do atributo _quantidade:768

Especificadores de acesso - static

- Métodos **static** também são métodos da classe
 - podem ser chamados mesmo se não houver objetos da classe criados
 - Não podem manipular atributos de instância nem chamar métodos de instância (por quê?)
- Exemplo: métodos da classe *Math*
 - podemos executar *Math.sin(double x)*, mesmo sem ter instanciado a classe *Math*.

Especificadores de acesso - final

- Especificador **final**
 - permite definir variáveis que não podem mais ter o seu valor modificado (constantes)
 - Exemplo:
final double PI = 3.1416;
 - Pode misturar especificadores:
static final double PI = 3.1416;

Especificadores de acesso - final

- Exemplos no DrJava
- Classe: ExemploFinal

```
class ExemploFinal
{
    static final double PI = 3.1416;
    static void imprimeValor()
    {
        System.out.println("Valor de PI:" + PI);
    }
}
```

```
> ExemploFinal.imprimeValor()
Valor de PI:3.1416
```

Especificadores de acesso - final

- Estilo: constantes são escritas com letras maiúsculas, usando hifens para separar as palavras se necessário:
 - final int NR_LINHAS = 30;

Exercício

Escreva uma classe Funcionario que tenha os atributos número funcional, nome, salário, porcentagemComissao, aliquotaImpostoRenda e qtdeFuncionariosCriados, com as seguintes especificações:

- os dados pessoais (número, nome e salário) devem ser fornecidos no momento da criação do objeto da classe;
- a alíquota do Imposto de renda é uma variável que não pode ser mudada após inicializada;
- o salário não pode ser alterado pelo usuário por meio de acesso direto;
- qtdeFuncionariosCriados é um atributo da classe e não específico de cada objeto;
- Crie os seguintes métodos (faça as considerações necessárias sobre tipos de atributos, métodos e especificadores de acesso):
 - alteraComissao (porcentagem) – altera porcentagem da comissão para o funcionário
 - calculaComissao() – calcula a Comissão – salário multiplicado pela porcentagem
 - calculaImpostoRenda() – calcula o IR: (salario + comissao) * alíquota
 - imprimeDados – imprime dados do funcionário.
 - main – cria três funcionários diferentes, inclui a porcentagem de comissão e calcula a comissão e IR de cada um deles, imprime todos os dados do funcionário e imprime a quantidade total de funcionário incluídos.

Construtores e Especificadores de Acesso

Professoras:
Fátima L. S. Nunes

