



Centro Tecnológico
Departamento de Informática

Prof. Vítor E. Silva Souza
<http://www.inf.ufes.br/~vitorsouza>

[Desenvolvimento OO com Java]
Organizando suas classes



Este obra está licenciada com uma licença Creative Commons Atribuição-
Compartilhagual 4.0 Internacional: <http://creativecommons.org/licenses/by-sa/4.0/>.

Conteúdo do curso

- O que é Java;
- Variáveis primitivas e controle de fluxo;
- Orientação a objetos básica;
- Um pouco de vetores;
- Modificadores de acesso e atributos de classe; 
- Herança, reescrita e polimorfismo;
- Classes abstratas e interfaces;
- Exceções e controle de erros;
- Organizando suas classes;
- Utilitários da API Java.

Estes slides foram baseados na [apostila do curso FJ-11: Java e Orientação a Objetos da Caelum](#) e na apostila Programação Orientada a Objetos em Java do [prof. Flávio Miguel Varejão](#).

Por que organizar as classes?

- À medida que aumenta o número de classes, aumenta a chance de **coincidência** de nomes;
- Precisamos separar as classes em **espaços** de nomes;
- Java possui o conceito de **pacotes**:
 - *Espaço de nome para evitar conflitos;*
 - *Agrupamento de classes semelhantes;*
 - *Maneira de construir bibliotecas de classes;*
 - *Estabelecimento de políticas de acesso às classes.*

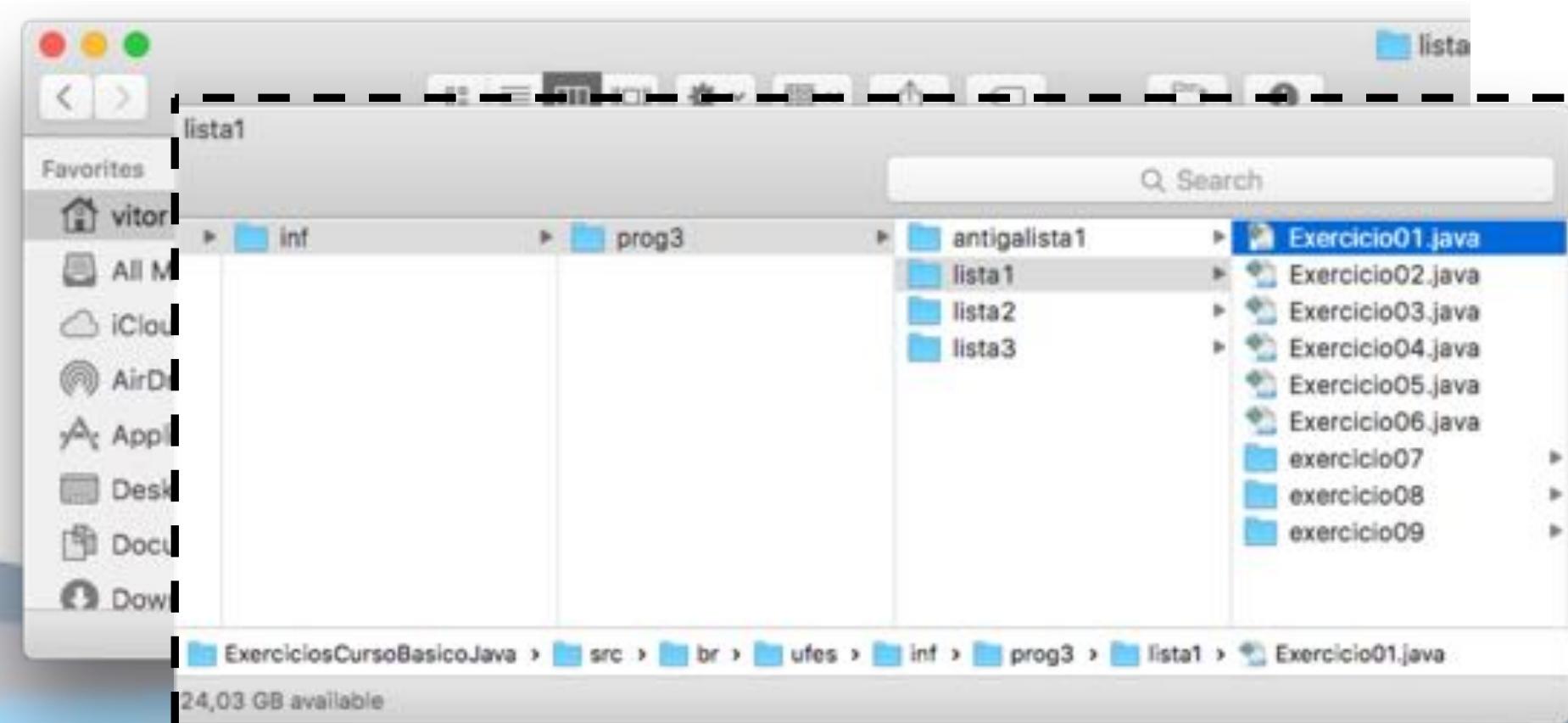


Pumpid	Date	pressure	temperature	volumeflow	rotational
1	2012-05-10	22222.0	33333.0	44444.0	2000.0
2	2012-05-11	22222.0	33333.0	44444.0	1500.0
3	2012-05-12	22222.0	33333.0	44444.0	2500.0
4	2012-05-13	22222.0	33333.0	44444.0	3000.0
5	2012-05-15	22222.0	33333.0	44444.0	2700.0
6	2012-05-04	22222.0	33333.0	44444.0	22.0
7	2012-05-11	22222.0	33333.0	44444.0	2000.0
8	2012-05-04	22222.0	33333.0	44444.0	5000.0
9	1970-01-01	22222.0	33333.0	44444.0	2000.0
10	1970-01-01	22222.0	33333.0	44444.0	1500.0

Inspiração: organização de arquivos

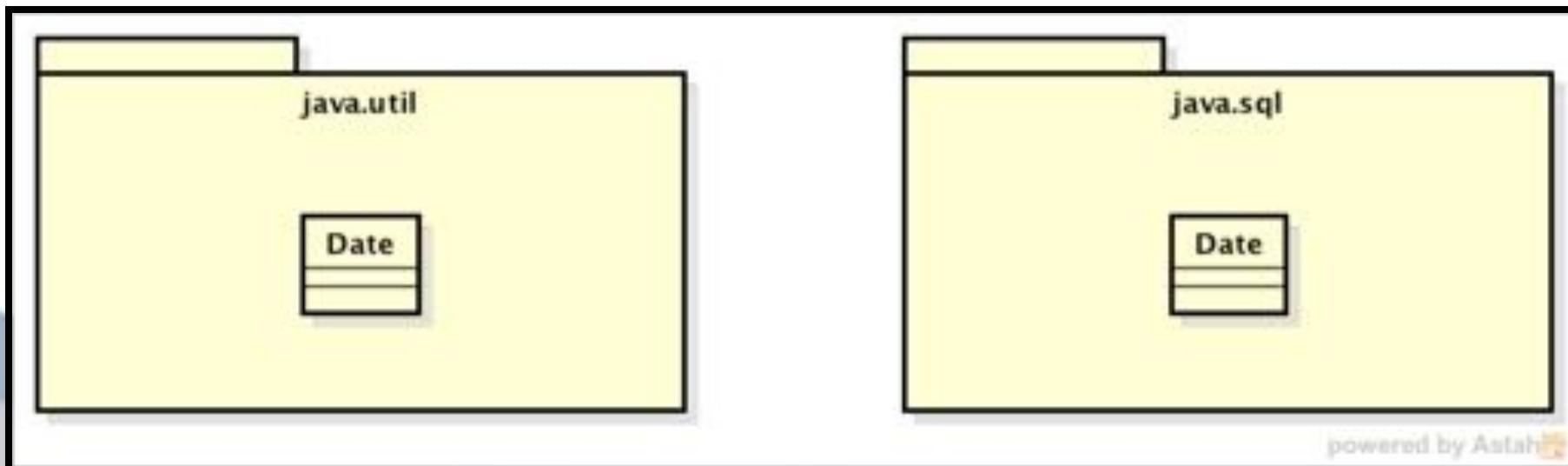
- Mesmo problema: não podemos ter 2 arquivos com mesmo nome na mesma pasta;
 - *Pastas definem espaços de nome, org. hierárquica.*

Em Java,
pacotes se
refletem em
pastas no
sistema



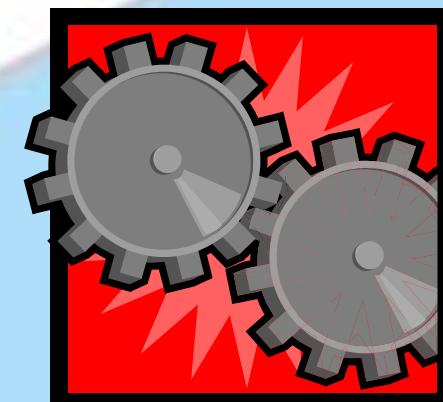
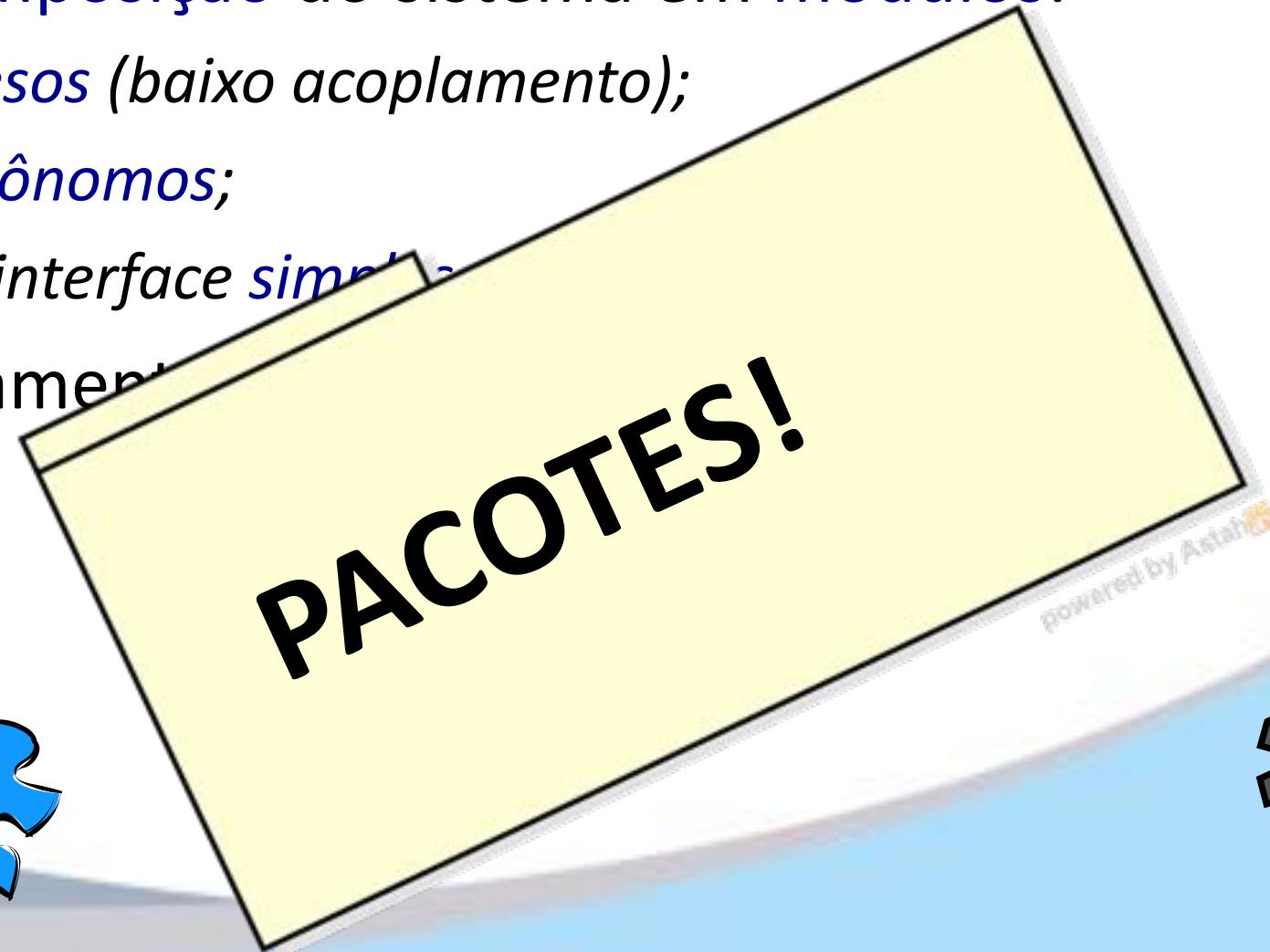
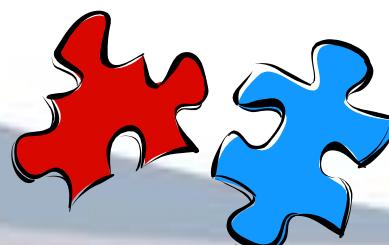
Pacotes da API Java

- As APIs Java (ex.: Java SE) são divididas em pacotes:
 - *java.lang*: classes do **núcleo** da plataforma;
 - *java.util*: classes **utilitárias**;
 - *java.io*: classes para **I/O** (entrada/saída);
 - *Dentre muitos outros...*

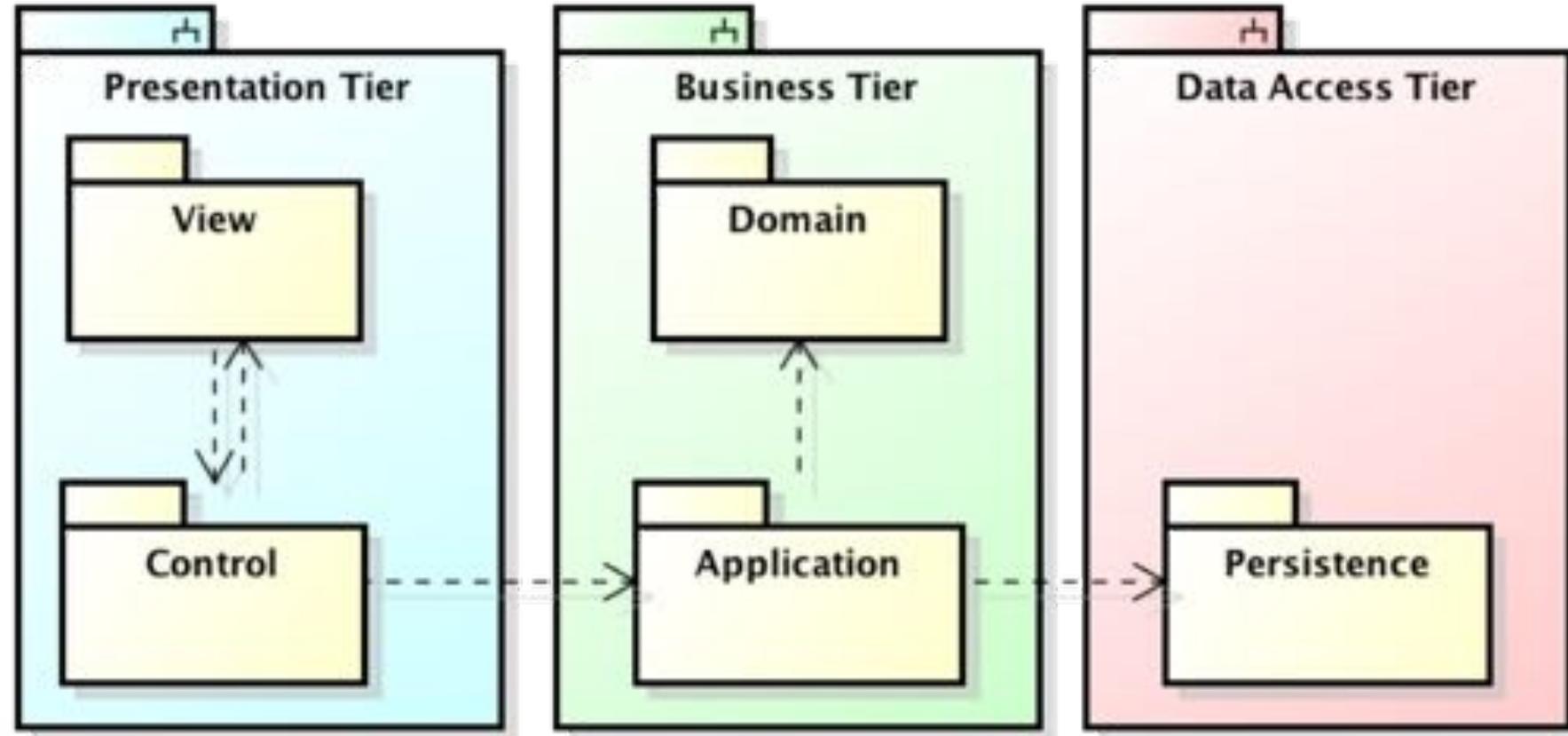


Modularidade

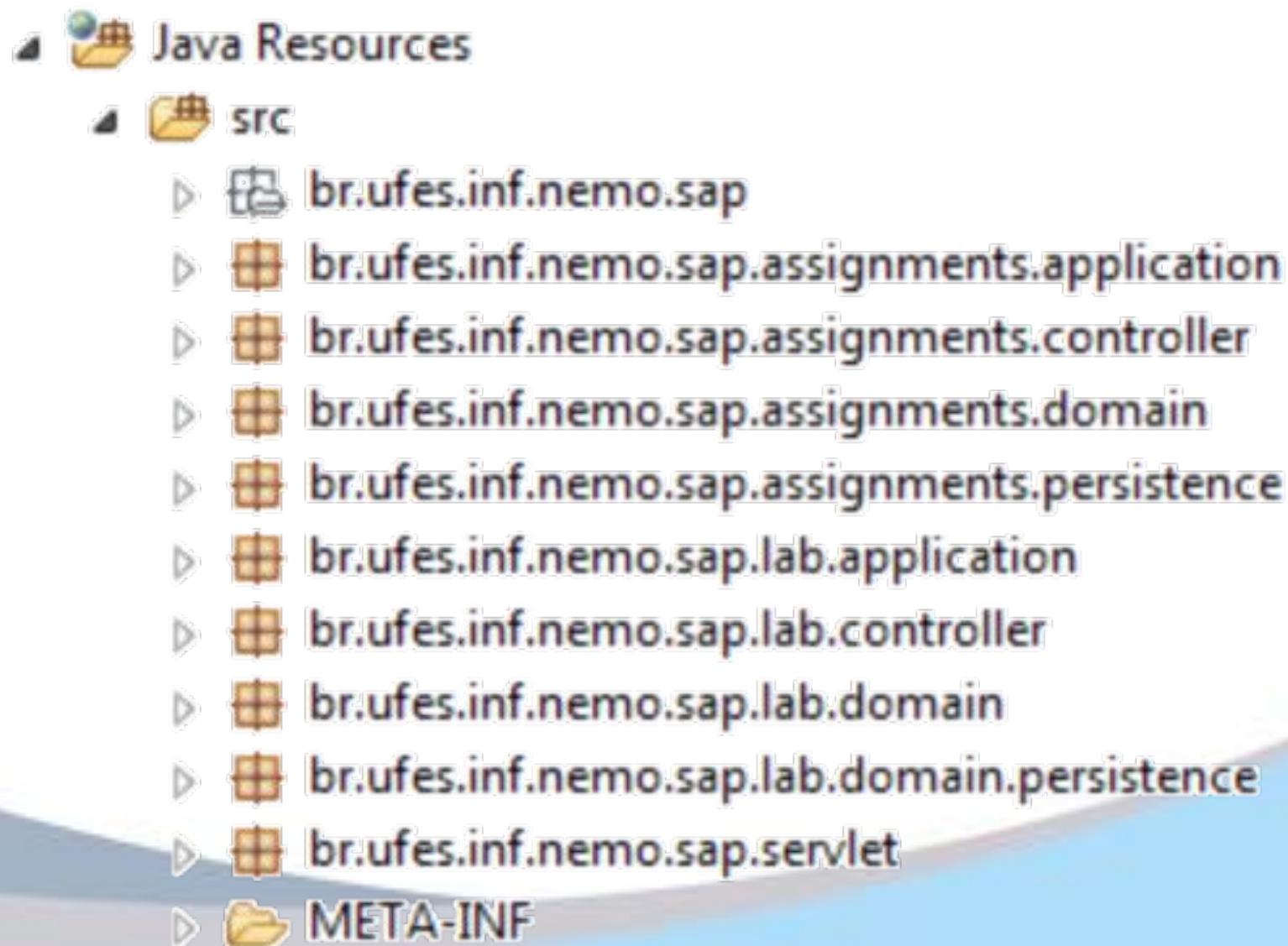
- Decomposição do sistema em módulos:
 - *Coesos* (*baixo acoplamento*);
 - *Autônomos*;
 - *De interface simples*;
- Fundamentais:



Exemplo: uma arquitetura para a Web



Exemplo: uma arquitetura para a Web



Declaração do pacote

- Uso da palavra-chave **package**;
- Primeira linha não comentada da classe:

```
package br.ufes.inf.prog3.lista1;
```

```
public class Exercicio01 { /* ... */ }
```

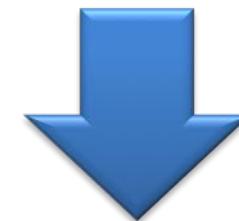
The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays a project named 'ExerciciosCursoBasicoJava' with a 'src' folder containing several packages: '(default package)', 'br.ufes.inf.prog3.antigalista1.exercicio08', 'br.ufes.inf.prog3.antigalista1.exercicio09', and 'br.ufes.inf.prog3.lista1'. Within 'br.ufes.inf.prog3.lista1', files 'Exercicio01.java', 'Exercicio02.java', 'Exercicio03.java', 'Exercicio04.java', 'Exercicio05.java', and 'Exercicio06.java' are listed, with 'Exercicio01.java' currently selected. On the right, the code editor shows the Java code for 'Exercicio01.java':

```
1 package br.ufes.inf.prog3.lista1;  
2  
3  
4 /*  
5 * Programa que calcula o valor das expressões  
6 *  
7 * @author Vitor E. Silva Souza (vitorsouza@gmail.com)  
8 * @version 1.0  
9 *  
10 public class Exercicio01 {
```

Convenção de nomes

- Para não haver **conflito** com absolutamente ninguém, sugere-se usar seu **domínio** na Internet ao contrário:

`http://nemo.inf.ufes.br`



`br.ufes.inf.nemo.sistema1`
`br.ufes.inf.nemo.sistema2`

- Usar apenas letras **minúsculas**;
- Esse **padrão** não se aplica à **API Java**.

Importação

- Acesso **direto** dentro do **mesmo** pacote:

```
package br.ufes.inf.prog3.lista1.exercicio07;
class Ponto { /* ... */ }
```

```
package br.ufes.inf.prog3.lista1.exercicio07;
class Triangulo {
    private Ponto vertice1;      /* ... */
}
```

```
package br.ufes.inf.prog3.lista1.exercicio07;
public class Exercicio07 {
    public static void main(String[] args) {
        Triangulo triangulo;
        /* ... */
    }
}
```

Importação

- O mesmo não ocorre em pacotes diferentes:

```
package br.ufes.inf.prog3.lista1.exercicio07.dominio;  
class Ponto { /* ... */ }
```

```
package br.ufes.inf.prog3.lista1.exercicio07.dominio;  
class Triangulo {  
    private Ponto vertice1;      /* ... */  
}
```

```
package br.ufes.inf.prog3.lista1.exercicio07;  
public class Exercicio07 {  
    public static void main(String[] args) {  
        Triangulo triangulo;  
        /* ... */  
    }  
}
```

error: Triangulo cannot be resolved to a type

Importação

- Resolve-se a questão importando a classe que encontra-se em outro pacote:

```
package br.ufes.inf.prog3.lista1.exercicio07;  
  
import br.ufes.inf.prog3.lista1.exercicio07.dominio.Triangulo;  
  
public class Exercicio07 {  
    public static void main(String[] args) {  
        Triangulo triangulo;  
        /* ... */  
    }  
}
```

Uma IDE ajuda nesta tarefa! Eclipse: "Organize Imports".

error: The type br.ufes.inf.prog3.lista1.exercicio07.
dominio.Triangulo is not visible

Importação

- A classe importada, no entanto, precisa ser pública!

```
package br.ufes.inf.prog3.lista1.exercicio07.dominio;
```

```
public class Triangulo {  
    private Ponto vertice1;      /* ... */  
}
```

```
package br.ufes.inf.prog3.lista1.exercicio07;
```

```
import br.ufes.inf.prog3.lista1.exercicio07.dominio.Triangulo;  
  
public class Exercicio07 {  
    public static void main(String[] args) {  
        Triangulo triangulo;  
        /* ... */  
    }  
}
```



Importação

- Pode-se importar classe por classe ou um **pacote inteiro**:

```
package br.ufes.inf.prog3.lista1.exercicio07;  
  
import br.ufes.inf.prog3.lista1.exercicio07.dominio.*;  
  
public class Exercicio07 {  
    public static void main(String[] args) {  
        Triangulo triangulo;  
        Ponto vertice1;  
        /* ... */  
    }  
}
```

Importação: e se der conflito?

- Se uma classe precisa **usar** outras duas classes de **mesmo nome**, só poderá **importar** uma delas:

```
package com.tables.tablesystem.gui;

import com.tables.tablesystem.guicomponents.Table;

public class ManageTablesWindow {
    public static void main(String[] args) {
        Table productTable;
        com.tables.tablesystem.domain.Table product;
        product = new com.tables.tablesystem.domain.Table();
        /* ... */
    }
}
```



A outra classe deverá ser referida pelo seu nome completo,
também conhecido como FQN (Fully Qualified Name).

Alguns detalhes

- Ordem das declarações num arquivo .java:
 - *package* [0..1];
 - *import* [0..*];
 - *class* [1..*];
- Importação de pacote inteiro (*import pacote.**):
 - *Não há perda de desempenho;*
 - *Pode haver problema de conflito de nomes;*
 - *Importar classe por classe é considerado boa prática, pois facilita a leitura;*
 - *"Organize Imports" do Eclipse faz assim por padrão.*

Uso do pacote `java.lang`

- As **classes** do pacote `java.lang` são importadas automaticamente;
- Não é necessário:
 - *import java.lang.String;*
 - *import java.lang.Math;*
 - *import java.lang.*;*

Importação estática

- A partir do Java 5 é possível importar os membros **estáticos** de uma classe:
- Antes:

```
/* ... */  
r = Math.exp(x) + Math.log(y) - Math.sqrt(Math.pow(Math.PI,  
y));
```

- Depois:

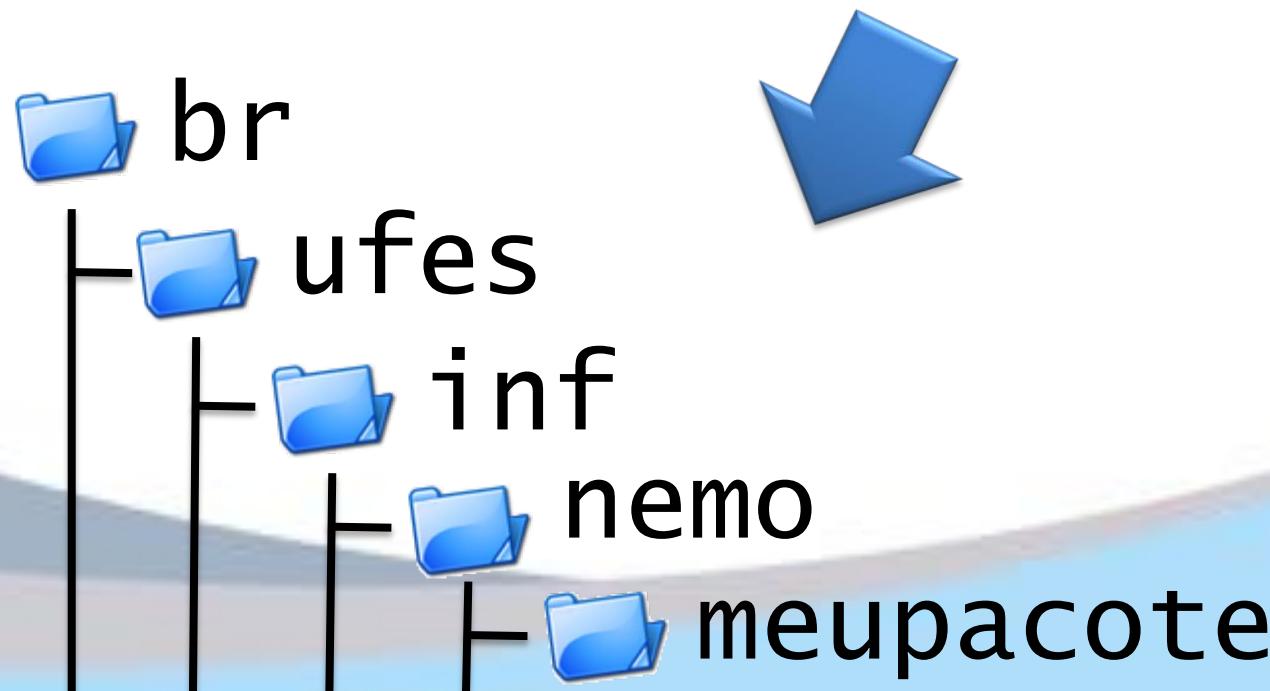
```
import static java.lang.Math.*;  
  
/* ... */  
r = exp(x) + log(y) - sqrt(pow(PI, y));
```

Também pode importar somente um específico.

Localização de pacotes

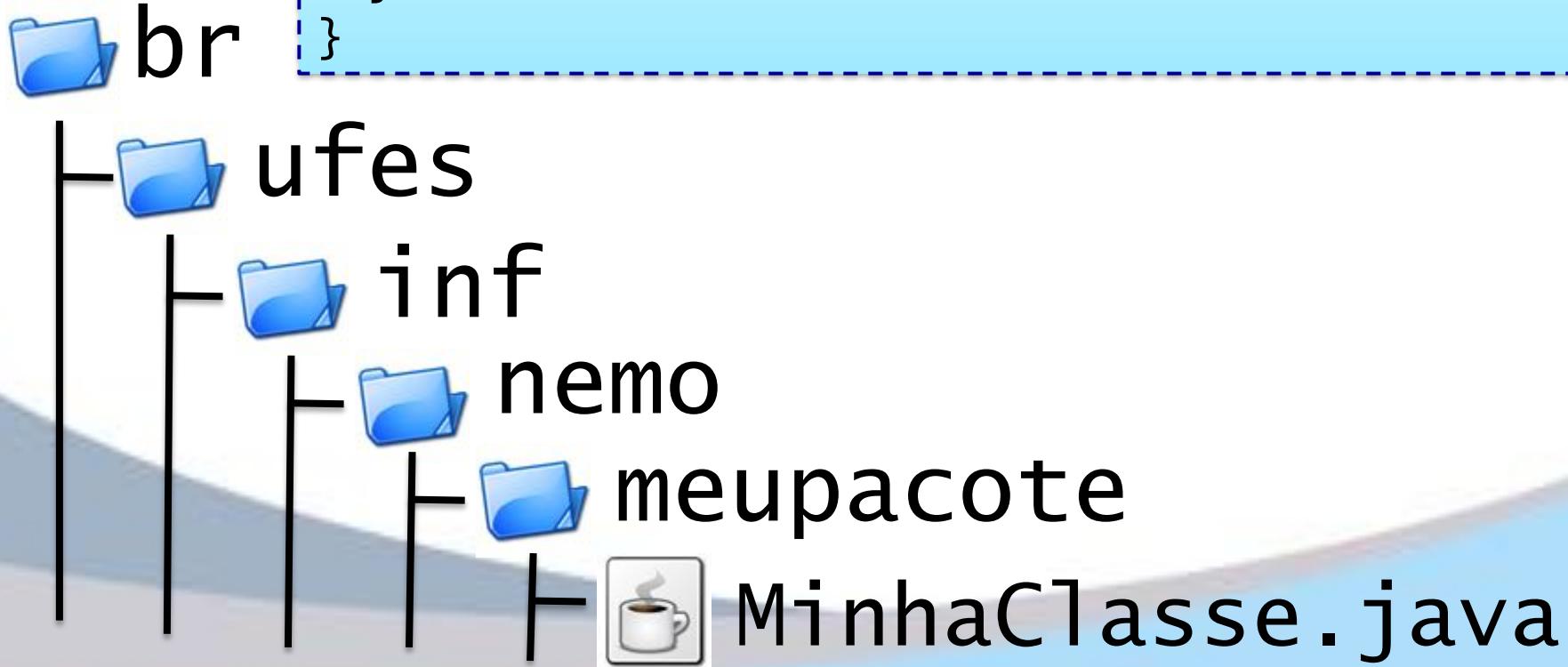
- A JVM **carrega classes** dos arquivos **.class**;
- Como a JVM **encontra** as classes em **diferentes pacotes**?

br.ufes.inf.nemo.meupacote



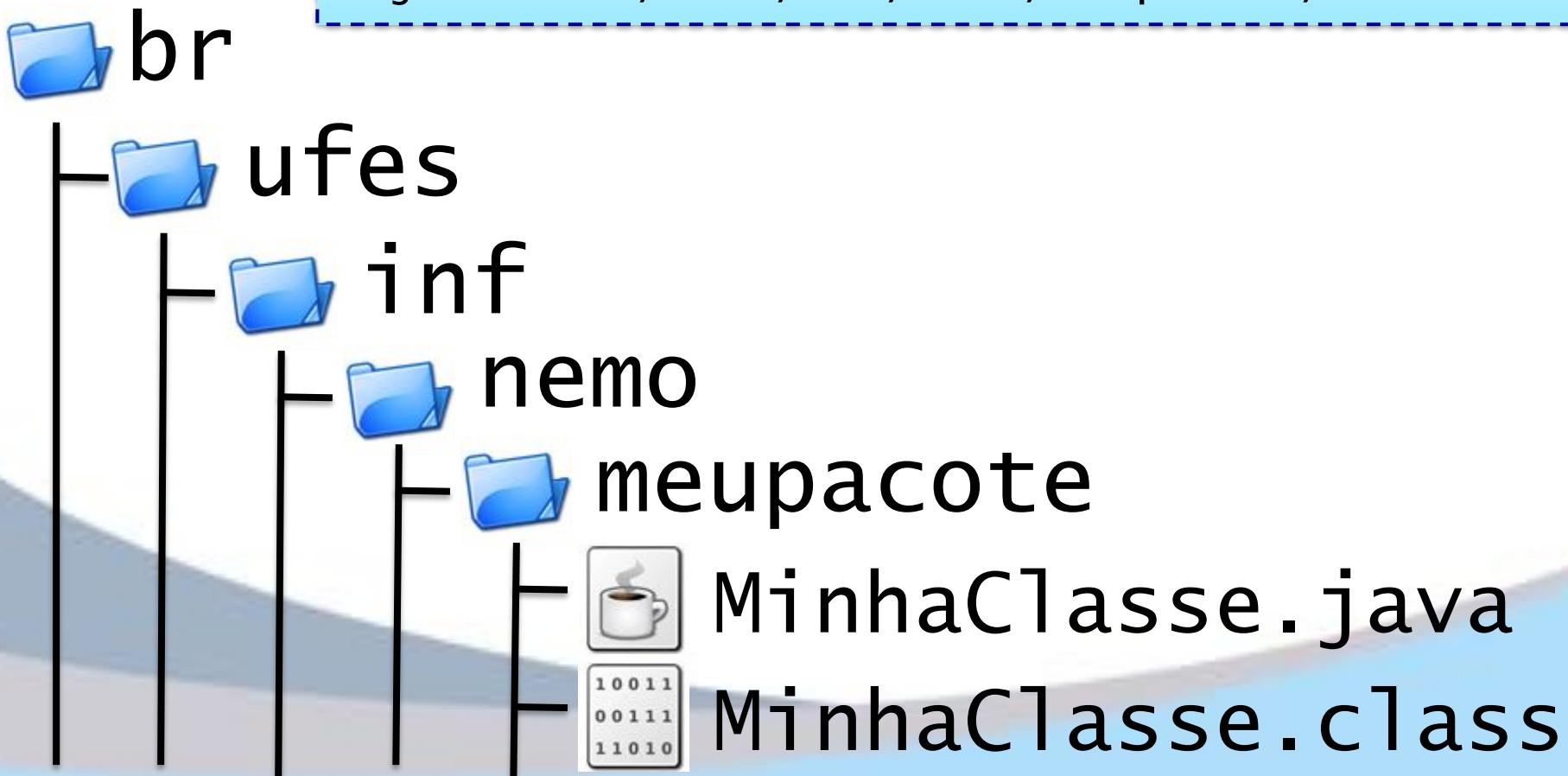
Localização de pacotes

```
package br.ufes.inf.nemo.meupacote;  
import java.util.Date;  
public class MinhaClasse {  
    public static void main(String[] args) {  
        System.out.println(new Date());  
    }  
}
```



Localização de pacotes

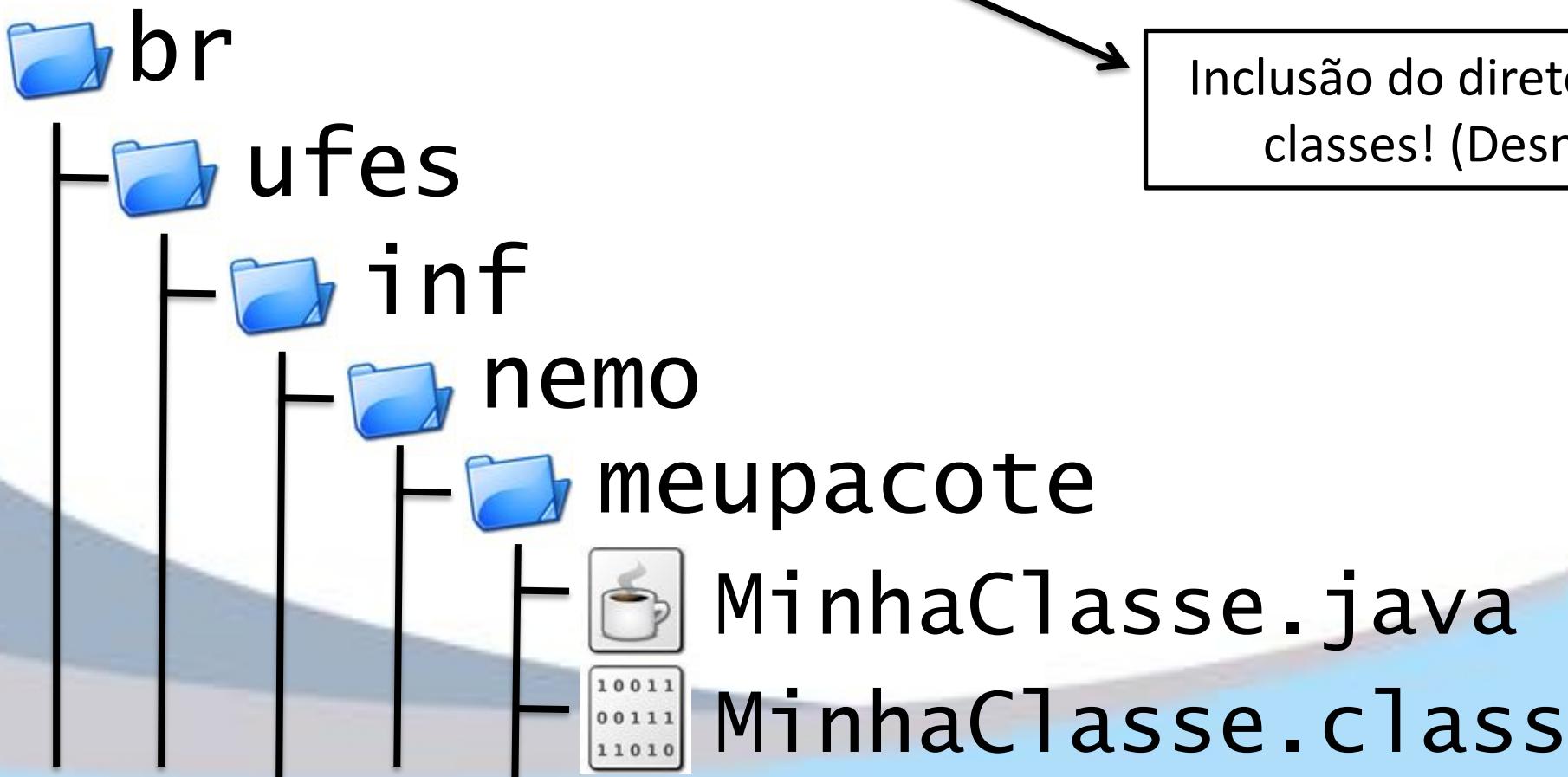
```
$ ls  
br  
  
$ javac br/ufes/inf/nemo/meupacote/MinhaClasse.java
```



Localização de pacotes

```
$ java -cp . br.ufes.inf.nemo.meupacote.MinhaClasse
```

```
Wed Jun 05 21:01:29 BRT 2013
```



Inclusão do diretório atual no caminho de classes! (Desnecessário no Java 5+)

Classpath

- O “caminho de classes” ou “trilha de classes” é onde as ferramentas do JDK e a JVM procuram classes;
 - A partir dos *diretórios do classpath* procura-se as classes segundo seus pacotes (usa a 1^a encontrada).
- Estão por padrão no *classpath*:
 - A biblioteca de classes da API Java SE;
 - O diretório *atual*.
- O *classpath* pode ser alterado:
 - Variável de ambiente (não recomendado);
 - Opção *-classpath* ou *-cp*.

Compilação automática

- Ao **compilar** uma classe, se ela faz referência a outra que não foi compilada, esta última é **compilada** se o código está **disponível**;
- Se já foi **compilada**, mas o arquivo fonte está com data mais **recente**, ela é recompilada.
- Uso de **IDEs**:
 - *Utilizar uma IDE abstrai todas estas preocupações;*
 - *A IDE cuida de todo o processo de compilação.*

O pacote padrão

- Toda classe que **não** especifica o pacote pertence ao pacote padrão;
- Seu `.class` deve estar numa pasta raiz do *classpath*.

```
public class Bolo {  
    public static void main(String[] args) {  
        // Não há import, estão no mesmo pacote.  
        Torta t = new Torta();  
        t.f();  
    }  
}  
  
class Torta {  
    void f() { System.out.println("Torta.f()"); }  
}
```

Especificadores de Acesso



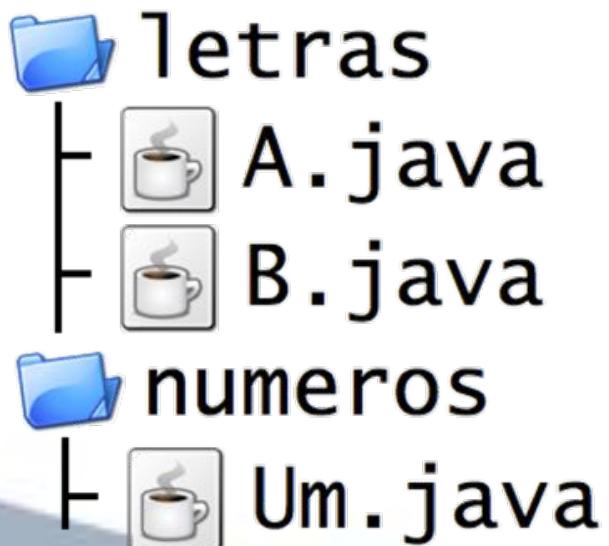
Membros públicos

Membro	Resultado
Classes	Classes públicas* podem ser importadas por qualquer classe.
Atributos	Atributos públicos podem ser lidos e alterados por qualquer classe.
Métodos	Métodos públicos podem ser chamados por qualquer classe.

* Só pode haver uma classe pública por arquivo-fonte e os nomes (da classe e do arquivo) devem ser iguais.

Membros públicos

```
public class A {  
    public int x = 10;  
    public void print() {  
        System.out.println(x);  
    }  
}
```



```
import letras.B;  
public class Um {  
    B b = new B();  
    public void g() {  
        b.f();  
    }  
}
```

```
public class B {  
    public A a = new A();  
    public void f() {  
        a.x = 15;  
        a.print();  
    }  
}
```

Finalmente, PSVM!

- O método main() é:
 - *public*, pois deve ser chamado pela **JVM**;
 - *static*, pois pertence à *classe* como um todo (a JVM não instancia um objeto para chamá-lo);
 - *void*, pois não retorna **nada**.
- A classe que possui o método main() deve ser:
 - *public*, pois deve ser acessível pela **JVM**.

Membros privados

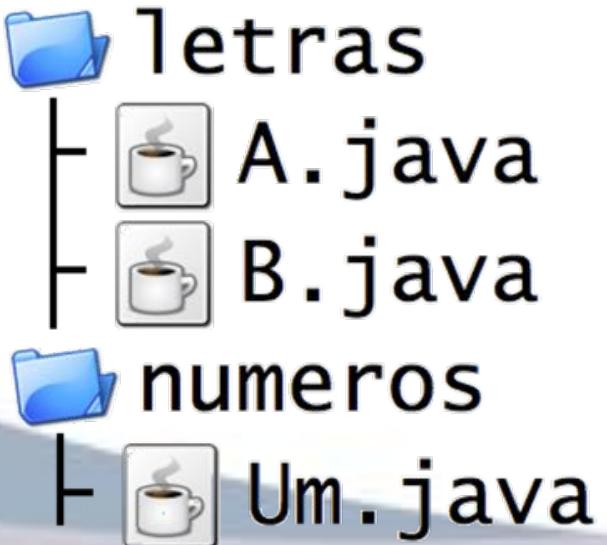
Membro	Resultado
Classes	Somente classes internas* podem ser declaradas privadas.
Atributos	Atributos privados só podem ser lidos e alterados pela própria classe.
Métodos	Métodos privados só podem ser chamados pela própria classe.

* Tópico avançado...

Membros privados

```
public class A {  
    private int x = 10;  
    private void print() {  
        System.out.println(x);  
    }  
    void incr() { x++; }  
}
```

```
import letras.B;  
public class Um {  
    B b = new B();  
    public void g() {  
        b.f();  
    }  
}
```



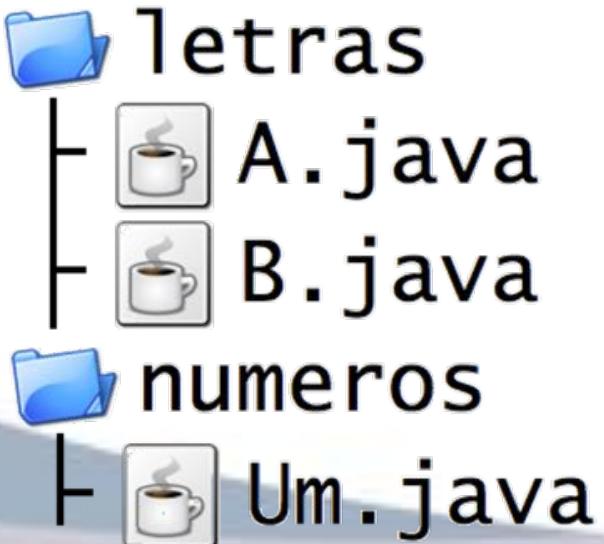
```
public class B {  
    public A a = new A();  
    public void f() {  
        // Erro: a.x = 15;  
        // Erro: a.print();  
    }  
}
```

Membros *package-private*

Membro	Resultado
Classes	Classes <i>package-private</i> só podem ser utilizadas por classes do mesmo pacote.
Atributos	Atributos <i>package-private</i> só podem ser lidos e alterados por classes do mesmo pacote.
Métodos	Métodos <i>package-private</i> só podem ser chamados por classes do mesmo pacote.

Membros *package-private*

```
class A {  
    int x = 10;  
    void print() {  
        System.out.println(x);  
    }  
    void incr() { x++; }  
}
```



```
import letras.*;  
public class Um {  
    // Erro: A a;  
    B b = new B();  
    public void g() {  
        // b.a.incr();  
        b.f();  
    }  
}
```

```
public class B {  
    A a = new A();  
    public void f() {  
        a.x = 15; a.print();  
    }  
}
```

Membros protegidos

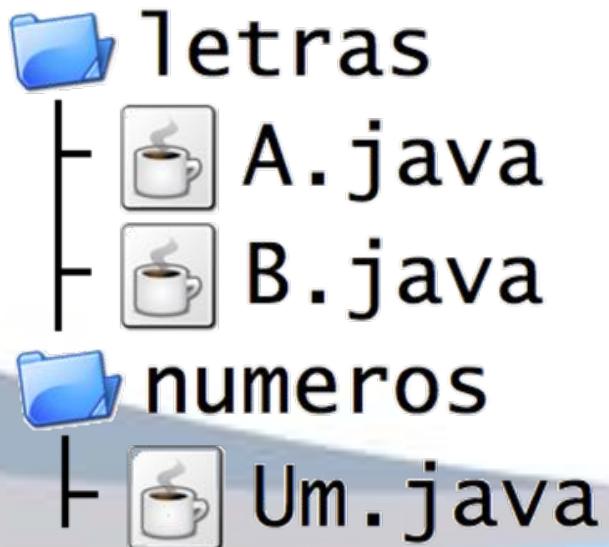
Membro	Resultado
Classes	Somente classes internas* podem ser declaradas protegidas.
Atributos	Atributos protegidos só podem ser lidos e alterados por classes do mesmo pacote ou subclasses*.
Métodos	Métodos protegidos só podem ser chamados por classes do mesmo pacote ou subclasses*.

* Tópico avançado...

Membros protegidos

```
public class A {  
    int x = 10;  
    protected void print() {  
        System.out.println(x);  
    }  
    protected void incr() {  
        x++; }  
}
```

```
import letras.*;  
public class Um extends A {  
    public void g() {  
        incr(); // OK!  
        print(); // OK!  
        // Erro: x++;  
    }  
}
```



```
public class B {  
    A a = new A();  
    public void f() {  
        a.x = 15; a.print();  
    }  
}
```

Modificadores de acesso

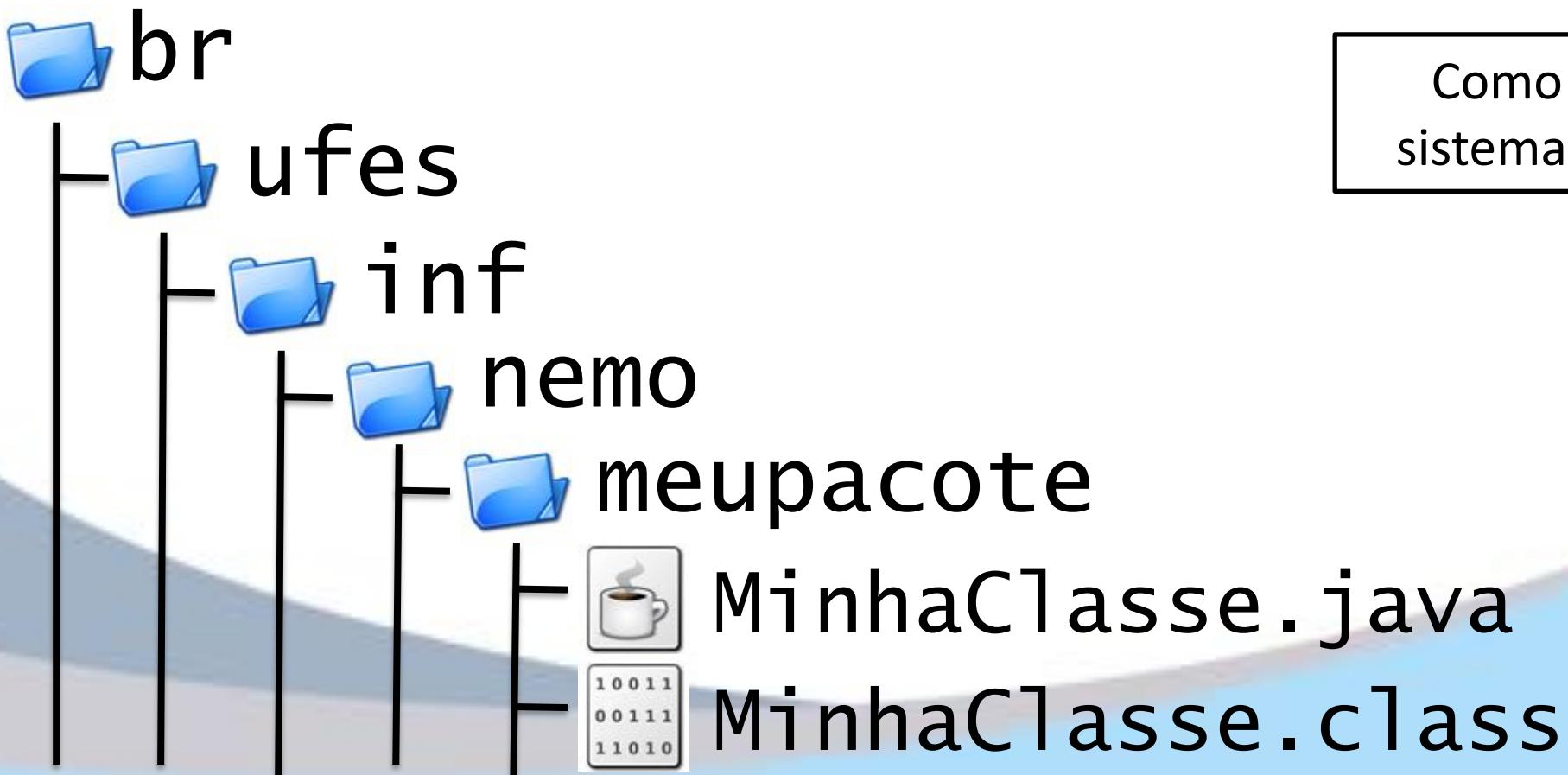
Acesso	Público	Protegido	P. Pacote	Privativo
A própria classe	Sim	Sim	Sim	Sim
Classe no mesmo pacote	Sim	Sim	Sim	Não
Subclasse em pacote diferente	Sim	Sim	Não	Não
Não-subclasse em pacote diferente	Sim	Não	Não	Não

Ferramentas jar e javadoc

Localização de pacotes

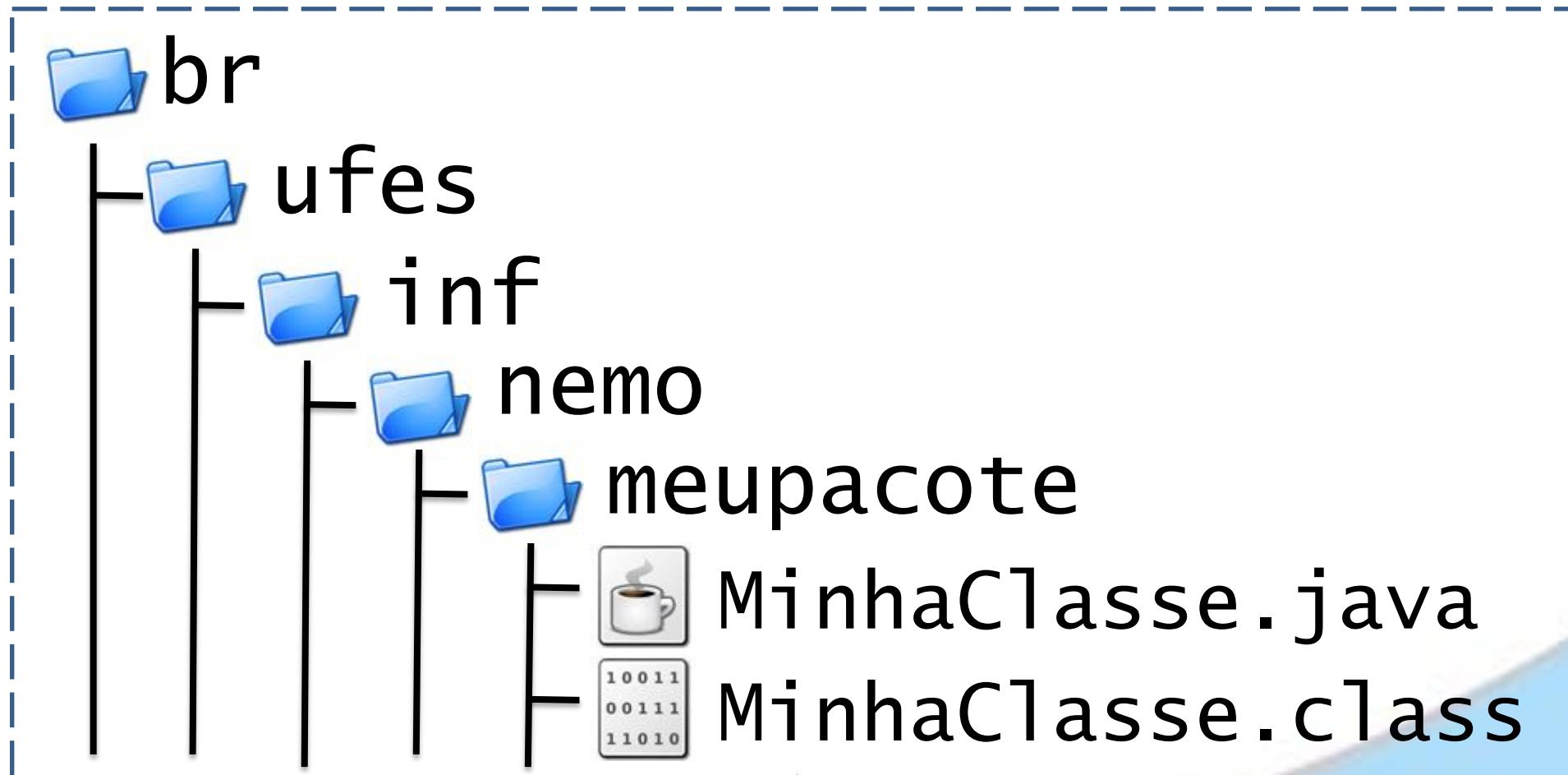
```
$ java -cp . br.ufes.inf.nemo.meupacote.MinhaClasse
```

```
Wed Jun 05 21:01:29 BRT 2013
```



Como disponibilizar esse sistema aos meus usuários?

Pacotes JAR



```
$ jar -c -f meujar.jar br/ufes/inf/nemo/meupacote/*.class
```



meujar.jar

Pacotes JAR

```
$ java -cp meujar.jar br.ufes.inf.nemo.meupacote.MinhaClasse
```

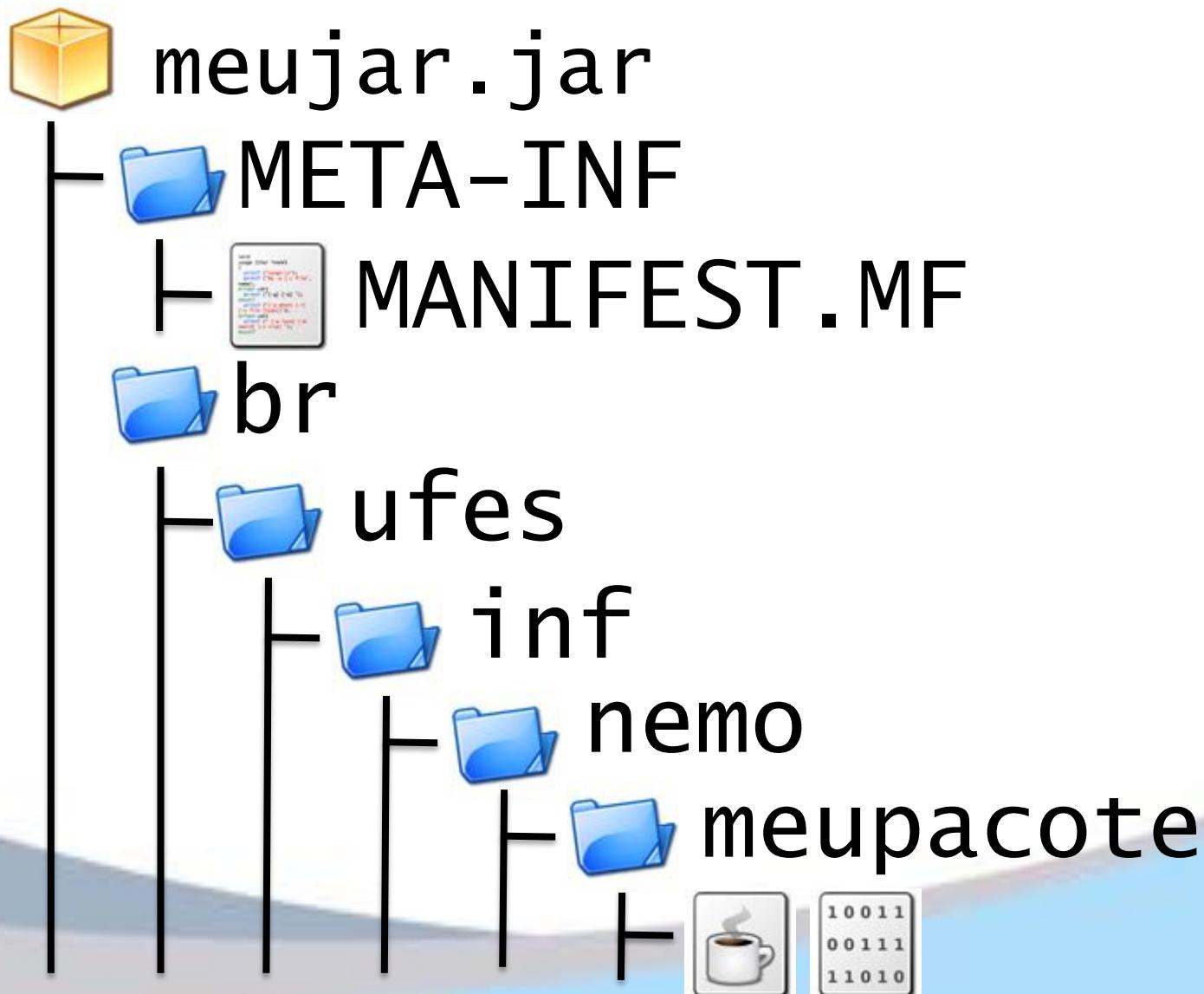
```
Wed Jun 05 21:15:06 BRT 2013
```



meujar.jar

Arquivos JAR são compactados no formato ZIP e podem ser abertos por qualquer programa compatível.

Pacotes JAR



O arquivo MANIFEST

- Contém meta-dados sobre o pacote:
- Crie um arquivo MANIFEST.MF:

```
Main-Class: br.ufes.inf.nemo.meupacote.MinhaClasse
```

- Digite os seguintes comandos:

```
$ jar -c -f meujar.jar -m MANIFEST.MF  
br/ufes/inf/nemo/meupacote/*.class
```

```
$ java -jar meujar.jar  
Wed Jun 05 21:23:03 BRT 2013
```

No Eclipse: File > Export > JAR File

JavaDoc

- Comentários são ignorados pelo compilador;
 - Usados pelo *programador* para melhorar a *legibilidade* do código;
 - Comentários de uma linha: `// ...;`
 - Comentários de múltiplas linhas: `/* ... */;`
- Um tipo, porém, é especial:
 - Comentários JavaDoc: `/** ... */` – utilizados pela ferramenta *javadoc* para criar uma *documentação HTML* das classes, atributos e métodos.
 - A *ferramenta javadoc* vem com o *JDK*;
 - Mais informações na apostila da Caelum.

JavaDoc: exemplo

```
/** <i>Documentação da classe</i>.  
 * @author Fulano da Silva  
 * @see java.io.File  
 */  
public class FileData extends File {  
    /** Documentação de atributo. */  
    private double tamanho;  
  
    /* Comentário  
       de múltiplas linhas. */  
  
    /** Documentação de método. */  
    public void excluir() {  
        int x = 1; // Comentário de uma linha.  
    }  
}
```

No Eclipse: Project > Generate Javadoc...

A documentação da API do Java

The screenshot shows a web browser displaying the Java API documentation for the `String` class. The URL is `http://ApplicationServer/java-10-8-api/api/index.html`. The page title is "Java™ Platform Standard Ed. 8". The navigation bar includes links for Overview, Package, Class, UML, Tree, Deprecated, Index, and Help. The "CLASS" tab is selected. Below the tabs are links for PREV CLASS, NEXT CLASS, FRAMES, and NO FRAMES. The main content area starts with the class hierarchy: `compact1 compact2 compact3`, followed by `java.lang`. The **Class String** section is shown, which extends `java.lang.Object` and implements `java.lang.String`. It lists implemented interfaces: `Serializable, CharSequence, Comparable<String>`. The class definition is provided in code:

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The text explains that the `String` class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class. It notes that strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because `String` objects are immutable they can be shared. For example:

```
String str = "abc";
```

This is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

At the bottom, it says: "Here are some more examples of how strings can be used."

Exercitar é fundamental

- Apostila FJ-11 da Caelum:
 - *Seção 12.6, página 167 (pacotes);*
 - *Seção 13.5, página 178 (JAR e JavaDoc).*