CAPÍTULO 12

Pacotes - Organizando suas classes e bibliotecas

"Uma discussão prolongada significa que ambas as partes estão erradas"
— Voltaire

Ao término desse capítulo, você será capaz de:

- separar suas classes em pacotes;
- preparar arquivos simples para distribuição.

12.1 - ORGANIZAÇÃO

Quando um programador utiliza as classes feitas por outro, surge um problema clássico: como escrever duas classes com o mesmo nome?

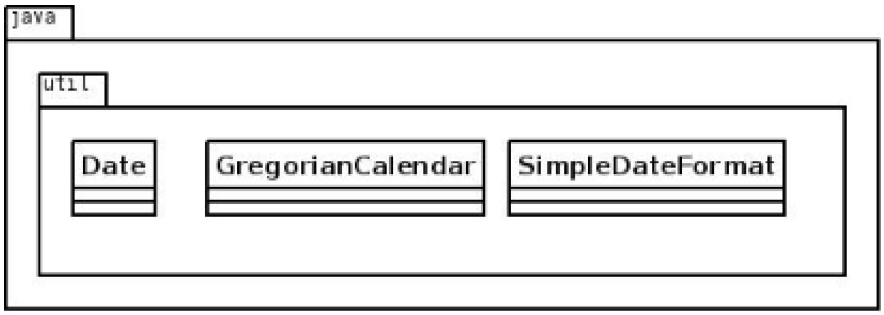
Por exemplo: pode ser que a minha classe de `Data` funcione de um certo jeito, e a classe `Data` de um colega, de outro jeito. Pode ser que a classe de `Data` de uma **biblioteca** funcione ainda de uma terceira maneira diferente.

Como permitir que tudo isso realmente funcione? Como controlar quem quer usar qual classe de `Data`?

Pensando um pouco mais, notamos a existência de um outro problema e da própria solução: o sistema operacional não permite a existência de dois arquivos com o mesmo nome sob o mesmo diretório, portanto precisamos organizar nossas classes em diretórios diferentes.

Os diretórios estão diretamente relacionados aos chamados **pacotes** e costumam agrupar classes de funcionalidades similares ou relacionadas.

Por exemplo, no pacote `java.util` temos as classes `Date`, `SimpleDateFormat` e `GregorianCalendar`; todas elas trabalham com datas de formas diferentes.



12.2 - Diretórios

Se a classe `Cliente` está no pacote `banco`, ela deverá estar no diretório com o mesmo nome: `banco`. Se ela se localiza no pacote `br.com.caelum.banco`, significa que está no diretório **`br/com/caelum/banco`**.

A classe `Cliente`, que se localiza nesse último diretório mencionado, deve ser escrita da seguinte forma:

```
package br.com.caelum.banco;

class Cliente {
  // ...
}
```

Fica fácil notar que a palavra chave `package` indica qual o pacote/diretório contém esta classe.

Um pacote pode conter nenhum ou mais subpacotes e/ou classes dentro dele.

Padrão da nomenclatura dos pacotes

O padrão da sun para dar nome aos pacotes é relativo ao nome da empresa que desenvolveu a classe:

```
br.com.nomedaempresa.nomedoprojeto.subpacote
br.com.nomedaempresa.nomedoprojeto.subpacote2
br.com.nomedaempresa.nomedoprojeto.subpacote2.subpacote3
```

Os pacotes só possuem letras minúsculas, não importa quantas palavras estejam contidas nele. Esse padrão existe para evitar ao máximo o conflito de pacotes de empresas diferentes.

As classes do pacote padrão de bibliotecas não seguem essa nomenclatura, que foi dada para bibliotecas de terceiros.

12.3 - Import

Para usar uma classe do mesmo pacote, basta fazer referência a ela como foi feito até agora simplesmente escrevendo o próprio nome da classe. Se quisermos que a classe `Banco` fique dentro do pacote `br.com.caelum.banco`, ela deve ser declarada assim:

```
package br.com.caelum.banco;

class Banco {
  String nome;
  Cliente clientes[];
}
```

Para a classe `Cliente` ficar no mesmo pacote, seguimos a mesma fórmula:

```
package br.com.caelum.banco;

class Cliente {
  String nome;
  String endereco;
}
```

A novidade chega ao tentar utilizar a classe `Banco` (ou `Cliente`) em uma outra classe que esteja fora desse pacote, por exemplo, no pacote `br.com.caelum.util`:

```
package br.com.caelum.banco.util;

class TesteDoBanco {

  public static void main(String args[]) {
    br.com.caelum.banco.Banco meuBanco = new br.com.caelum.banco.Banco();
```

```
    meuBanco.nome = "Banco do Brasil";
    System.out.println(meuBanco.nome);
  }

}
```

Repare que precisamos referenciar a classe `Banco` com todo o nome do pacote na sua frente. Esse é o conhecido *Fully Qualified Name* de uma classe. Em outras palavras, esse é o verdadeiro nome de uma classe, por isso duas classes com o mesmo nome em pacotes diferentes não conflitam.

Mesmo assim, ao tentar compilar a classe anterior, surge um erro reclamando que a classe `Banco` não está visível.

Acontece que as classes só são visíveis para outras no **mesmo pacote** e, para permitir que a classe `TesteDoBanco` veja e acesse a classe `Banco` em outro pacote, precisamos alterar essa última e transformá-la em pública:

```
package br.com.caelum.banco;

public class Banco {
  String nome;
  Cliente clientes[] = new Cliente[2];
}
```

A palavra chave `public` libera o acesso para classes de outros pacotes. Do mesmo jeito que o compilador reclamou que a classe não estava visível, ele reclama que o atributo/variável membro também não está. É fácil deduzir como resolver o problema: utilizando novamente o modificador `public`:

```
package br.com.caelum.banco;

public class Banco {
  public String nome;
  public Cliente clientes[] = new Cliente[2];
}
```

Podemos testar nosso exemplo anterior, lembrando que utilizar atributos como público não traz encapsulamento e está aqui como ilustração.

Voltando ao código do `TesteDoBanco`, é necessário escrever todo o pacote para identificar qual classe queremos usar? O exemplo que usamos ficou bem complicado de ler:

```
br.com.caelum.banco.Banco meuBanco = new br.com.caelum.banco.Banco();
```

Existe uma maneira mais simples de se referenciar a classe `Banco`: basta **importá-la** do pacote `br.com.caelum.banco`:

```
package br.com.caelum.banco.util;

// para podermos referenciar
// a Banco diretamente
import br.com.caelum.banco.Banco;

class TesteDoBanco {

  public static void main(String args[]) {
    Banco meuBanco = new Banco();
    meuBanco.nome = "Banco do Brasil";
  }

}
```

Isso faz com que não precisemos nos referenciar utilizando o *fully qualified name*, podendo utilizar `Banco` dentro do nosso código em vez de escrever o longo `br.com.caelum.banco.Banco`.

package, import, class

É muito importante manter a ordem! Primeiro, aparece uma (ou nenhuma) vez o `package`; depois, pode aparecer um ou mais `imports`; e, por último, as declarações de classes.

import x.y.z.*;

É possível "importar um pacote inteiro" (todas as classes do pacote, **exceto os subpacotes**) através do coringa *:

```
import java.util.*;
```

Importar todas as classes de um pacote não implica em perda de performance em tempo de execução, mas pode trazer problemas com classes de mesmo nome! Além disso, importar de um em um é considerado boa prática, pois facilita a leitura para outros programadores. Uma IDE como o Eclipse já vai fazer isso por você, assim como a organização em diretórios.

12.4 - Acesso aos atributos, construtores e métodos

Os modificadores de acesso existentes em Java são quatro, e até o momento já vimos três,

mas só explicamos dois.

- `public` - Todos podem acessar aquilo que for definido como `public`. Classes, atributos, construtores e métodos podem ser `public`.

- `protected` - Aquilo que é `protected` pode ser acessado por todas as classes do mesmo pacote e por todas as classes que o estendam, mesmo que essas não estejam no mesmo pacote. Somente atributos, construtores e métodos podem ser `protected`.

- **padrão (sem nenhum modificador)** - Se nenhum modificador for utilizado, todas as classes do mesmo pacote têm acesso ao atributo, construtor, método ou classe.

- `private` - A única classe capaz de acessar os atributos, construtores e métodos privados é a própria classe. Classes, como conhecemos, não podem ser `private`, mas atributos, construtores e métodos sim.

Classes públicas

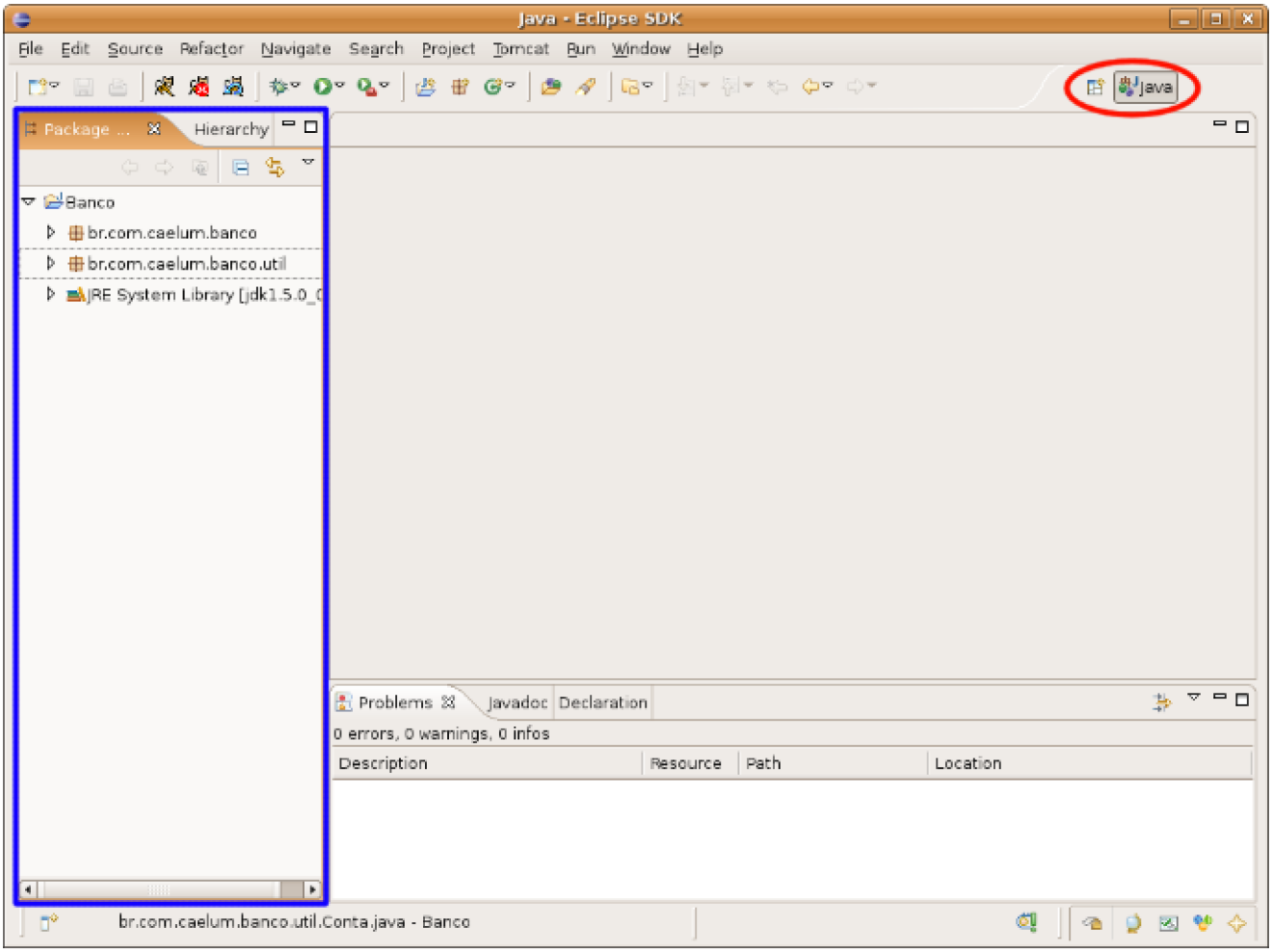
Para melhor organizar seu código, o Java não permite mais de uma classe pública por arquivo e o arquivo deve ser `NomeDaClasse.java`.

Uma vez que outros programadores irão utilizar essa classe, quando precisarem olhar o código da mesma, fica mais fácil encontrá-la sabendo que ela está no arquivo de mesmo nome.

Classes aninhadas podem ser `protected` ou `private`, mas esse é um tópico avançado que não será estudado nesse momento.

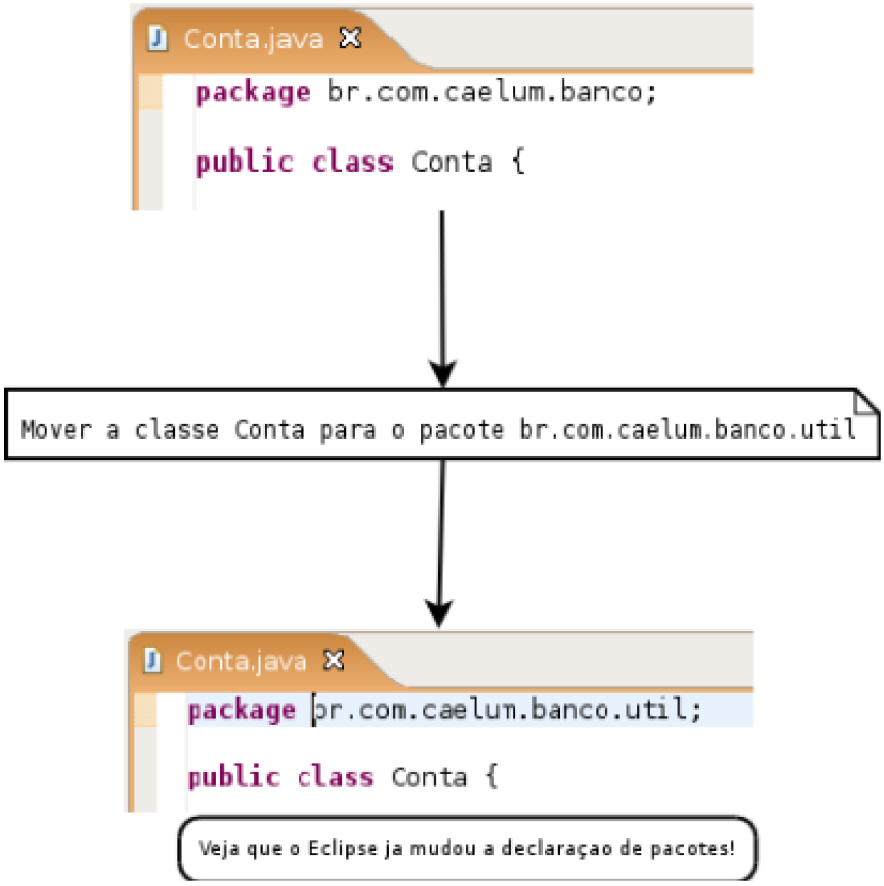
12.5 - Usando o Eclipse com pacotes

Você pode usar a perspectiva Java do Eclipse. A view principal de navegação é o *Package Explorer*, que agrupa classes pelos pacotes em vez de diretórios (você pode usá-la em conjunto com a *Navigator*, basta também abri-la pelo *Window/Show View/Package Explorer*).



Antes de movermos nossas classes, declare-as como públicas e coloque-as em seus respectivos arquivos: um arquivo para cada classe.

Você pode mover uma classe de pacote arrastando-a para o destino desejado. Repare que o Eclipse já declara `packages` e `imports` necessários:



No Eclipse nunca precisamos declarar um `import`, pois ele sempre vai sugerir isso quando usarmos o `Ctrl+Espaço` no nome de uma classe.

Você também pode usar o `Ctrl+1` no caso da declaração de pacote possuir algum erro.

12.6 - Exercícios: Pacotes

Atenção: utilize os recursos do Eclipse para realizar essas mudanças. Use a `view package-explorer`, que vai auxiliar bastante a manipulação dos arquivos e diretórios. Também utilize os quick fixes quando o Eclipse reclamar dos diversos problemas de

compilação que aparecerão. É possível fazer esse exercício inteiro **sem modificar uma linha de código manualmente**. Aproveite para praticar e descobrir o Eclipse, evite usá-lo apenas como um editor de texto.

Por exemplo, com o Eclipse nunca precisamos nos preocupar com os imports: ao usar o auto complete, ele já joga o import lá em cima. E, se você não fez isso, ele sugere colocar o `import`.

1. Selecionando o `src` do seu projeto, faça **ctrl + N** e escreva `Package`, ponha o seu sistema de Contas para utilizar pacotes. Na janela de criação de pacotes escreva o nome completo, desde o br, e o Eclipse tratará de fazer a separação das pastas corretamente.

Respeite a convenção de código da Sun, por exemplo:

- `br.com.empresa.banco`: colocar classes com o método `main` aqui (os Testes)
- `br.com.empresa.banco.conta` : colocar `Conta`, suas filhas e exceptions aqui
- `br.com.empresa.banco.sistema` : colocar `AtualizadorDeContas` aqui

Antes de corrigir qualquer erro de compilação, primeiro **mova todas as suas classes**, sem deixar nenhuma no pacote *default*.

2. Se você ainda não tiver separado cada classe em um arquivo, essa é a hora de mudar isso. Coloque cada classe em seu respectivo arquivo `.java`. Faça isso independente de ela ser pública: é uma boa prática.

3. O código não vai compilar prontamente, pois muitos métodos que declaramos são *package-private* quando, na verdade, precisaríamos que eles fossem `public`.

O mesmo vale para as classes: algumas delas precisarão ser públicas.

Use o recurso de quick fix do Eclipse aqui: ele mesmo vai sugerir que o modificador de acesso deve ser público. Para isso, use o **ctrl + 1** em cada um dos erros, escolhendo o *quick fix* mais adequado para seu problema.

4. (Opcional) Abra a view `Navigator` para ver como ficou os arquivos no sistema de arquivos do seu sistema operacional. Para isso, use **ctrl + 3**, comece a digitar `Navigator` e escolha a opção de abrir essa view..

CAPÍTULO ANTERIOR:

Exceções e controle de erros

PRÓXIMO CAPÍTULO:

Ferramentas: jar e javadoc

Twitter