



11-Organizando Classes em Pacotes

Prof. Marcos L. Chaim
EACH / USP Leste

ACH-2003 – Computação Orientada a
Objetos

1o. Semestre / 2007

Slides elaborados pelo Prof. Ivandré Paraboni



Pacotes (packages)

Conjuntos de classes relacionadas são organizadas em pacotes para:

- facilitar a localização e uso de tipos;

- evitar conflitos de nomes;

- fazer controle de acesso.



O que é um pacote ?

Um pacote é uma coleção de classes relacionadas provendo acesso protegido e gerenciamento de espaço de nomes.



Pacotes da plataforma Java

Os tipos nativos da plataforma Java são membros de vários pacotes que agrupam classes por função;

Por exemplo, classes fundamentais se encontram em `java.lang` e classes de E/S estão em `java.io`;

O programador também pode agrupar suas próprias classes em pacotes.



Exemplo

Considere um grupo de classes para representar uma coleção de objetos gráficos (`Circle`, `Rectangle` etc) e uma interface (`Draggable`) a ser implementada pelas classes que podem ser arrastadas com uso do mouse.



Exemplo (cont.)

```
//no arquivo Graphic.java
public abstract class Graphic {
    . . .
}

// no arquivo Circle.java
public class Circle extends Graphic implements Draggable {
    . . .
}

// no arquivo Rectangle.java
public class Rectangle extends Graphic implements Draggable {
    . . .
}

// no arquivo Draggable.java
public interface Draggable {
    . . .
}
```



Por que agrupar classes em pacotes ?

Para que você e outros programadores possam determinar facilmente que estes tipos são relacionados;

Para que os nomes de seus tipos não entrem em conflito com nomes de tipos de outros pacotes

cada pacote cria seu próprio espaço de nomes);

Para que os tipos dentro de seu pacote possam acessar uns aos outros de forma irrestrita, porém restringindo o acesso a tipos de outros pacotes.



Criando um pacote

Para criar um pacote, coloque tipos (classes, interfaces, etc) dentro dele;
A primeira linha de cada arquivo-fonte deve conter o comando *package* seguido do nome do pacote.



Criando um pacote - exemplo

```
// no arquivo Circle.java  
  
package graphics;  
  
public class Circle extends Graphic implements Draggable {  
    . . .  
}
```

A classe `Circle` é um membro público do pacote `graphics`.



Criando um pacote (cont.)

Inclua um comando `package` no início de cada arquivo-fonte que deva ser um membro daquele pacote, e.g.:

```
// no arquivo Rectangle.java  
  
package graphics;  
  
public class Rectangle extends Graphic implements Draggable {  
    . . .  
}
```



Escopo de pacotes

O escopo do comando `package` é o arquivo-fonte inteiro;

Por exemplo, todas as classes, interfaces etc. definidos em `Circle.java` são também membros do pacote `graphics`.

Havendo múltiplas classes em um mesmo arquivo, somente uma pode ser `public`, e deve ter o mesmo nome do arquivo-fonte.

Somente os membros públicos de um pacote são visíveis ao meio externo.



Relembrando: visibilidade

`public` – o item em questão é visível a outras classes etc;

`private` – visível apenas aos componentes da classe atual;

`protected` – visível somente a classe atual e seus descendentes.



O pacote “default”

Se nenhum nome de pacote for utilizado, seus tipos serão membros de um pacote `default`, que é um pacote sem nome;

Esta prática só faz sentido em aplicações muito pequenas, de caráter temporário, ou em uma fase muito incipiente da programação.



Nomeando um pacote

Com programadores Java do mundo todo escrevendo classes, interfaces etc, é provável que um mesmo nome seja dado a classes diferentes...

Exemplo: a classe `Rectangle` já existe no pacote `java.awt`;

No entanto, o compilador permite esta duplicidade.

Por quê ?



Espaços de nomes

As duas classes `Rectangle` do exemplo estão em pacotes distintos, e o nome completo de uma classe inclui o nome de seu pacote:

`graphics.Rectangle`

`java.awt.Rectangle`

Mas e se dois programadores usarem o mesmo nome para seus pacotes ?



Convenção para nomes de pacotes

Companhias usam seus nomes de domínio da Internet em ordem reversa para nomear seus pacotes, e.g.:

`br.com.companhia.pacote`

Conflitos de nomes só precisam ser resolvidos por convenção dentro da própria companhia, e.g.,:

`br.com.companhia.região.pacote`



Nomes de pacotes inválidos

Se o domínio Internet contém hifens, palavras Java reservadas, ou inicia por um dígito etc, usa-se o caractere de sublinhado “_”:

Domínio: **Pacote:**

<code>clipart-open.org</code>	<code>org.clipart_open</code>
<code>free.fonts.int</code>	<code>int_.fonts.free</code>
<code>poetry.7days.com</code>	<code>com._7days.poetry</code>



Usando membros de um pacote

Apenas membros `public` de um pacote são visíveis fora do pacote no qual foram definidos;

O acesso pode ser feito de 3 formas:

1. Fazendo referência ao nome completo;
2. Importando o membro de seu pacote;
3. Importando o pacote inteiro.



1-Usando o nome completo

Se o código que está sendo escrito pertence ao mesmo pacote que contém o membro em questão, basta usar seu nome simples, e.g.:

`Rectangle.`

Se o membro pertence a outro pacote, podemos usar seu nome completo , e.g.: `graphics.Rectangle.`



1-Usando o nome completo

O nome completo pode ser usado normalmente em qualquer referência ao membro em questão:

```
graphics.Rectangle minhaReta = new  
graphics.Rectangle();
```

Mas e se precisarmos referenciar este mesmo membro muitas vezes ?



2-Importando um membro de um pacote

Para importar um membro específico de um pacote usamos `import` logo depois da definição do pacote:

```
import graphics.Rectangle
```

`Rectangle` pode então ser referenciado normalmente pelo seu nome simples:

```
Rectangle meuRetângulo = new Rectangle();
```

Mas e se precisarmos de muitos membros de um mesmo pacote ?



3-Importando um pacote inteiro

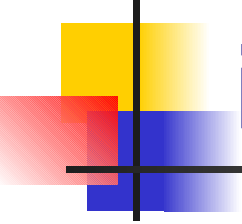
Usa-se o comando `import.*`

```
import graphics.*;
```

Assim qualquer classe do pacote `graphics` pode ser referenciada pelo seu nome simples;

Mas o caractere `*` não pode ser usado para representar substrings !

```
import graphics.A*;    // não funciona !!!
```



3-Importando um pacote inteiro (cont.)

O compilador Java importa automaticamente três pacotes inteiros:

1. O pacote `default` (sem nome);
2. O pacote `java.lang`;
3. O pacote atual.



Observação: hierarquia de pacotes

Pacotes não são hierárquicos !

Importando `java.util.*` não significa que podemos referenciar a classe

`Pattern` como `regex.Pattern`.

É preciso referenciá-la como:

`java.util.regex.Pattern` ou

(se importamos `java.util.regex.*`)
simplesmente como `Pattern`.



Resolvendo ambigüidades

E se importamos dois pacotes que possuem classes com o mesmo nome ?
Cada classe precisa ser referenciada pelo seu nome completo, e.g.:

```
import graphics;  
import java.awt;  
  
Rectangle meuRetângulo;           // ambíguo  
  
graphics.Rectangle meuRetângulo; // correto
```



Gerenciamento de arquivos-fonte e de classes

Muitas implementações da plataforma Java tiram proveito da estrutura hierárquica do sistema de arquivos para organizar arquivos-fonte e de classes;



Gerenciamento de arquivos-fonte e de classes (cont.)

Como funciona:

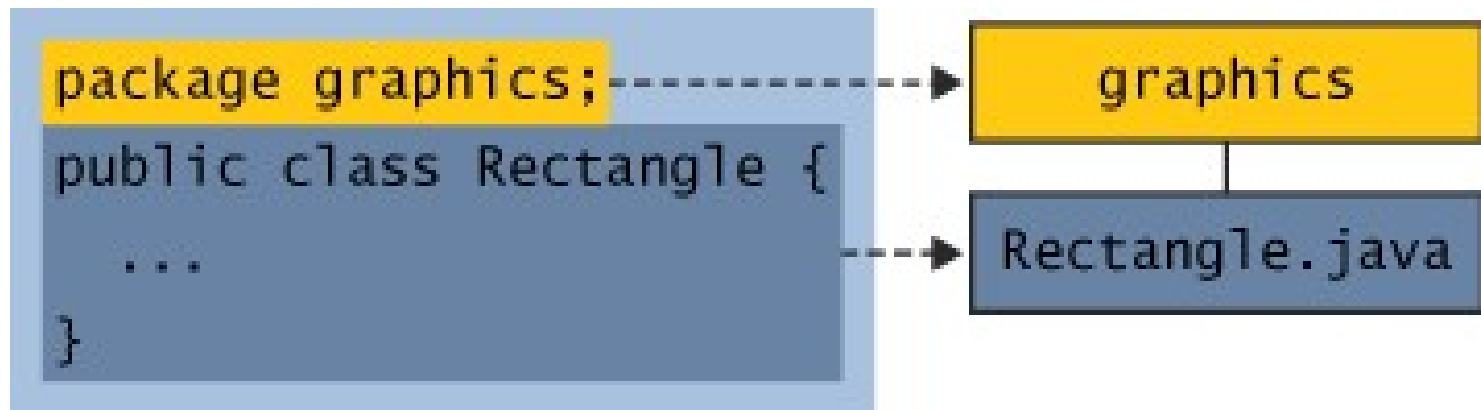
O código-fonte de uma classe etc. é armazenado em um arquivo texto cujo nome é o próprio nome do tipo, e cuja extensão é `.java`;

Este arquivo é colocado em um diretório de nome igual ao do pacote ao qual ele pertence;

Exemplo: o código-fonte da classe `Rectangle` seria armazenado em um arquivo

`Rectangle.java` em um diretório `graphics` (que pode estar em qualquer parte do sistema de arquivos).

Gerenciamento de arquivos-fonte e de classes (cont.)



Em um sistema de arquivos tipo MS-Windows a estrutura de nomes reflete a estrutura de diretórios:

Nome da classe: `graphics.Rectangle`

Caminho ao arquivo: `graphics\Rectangle.java`



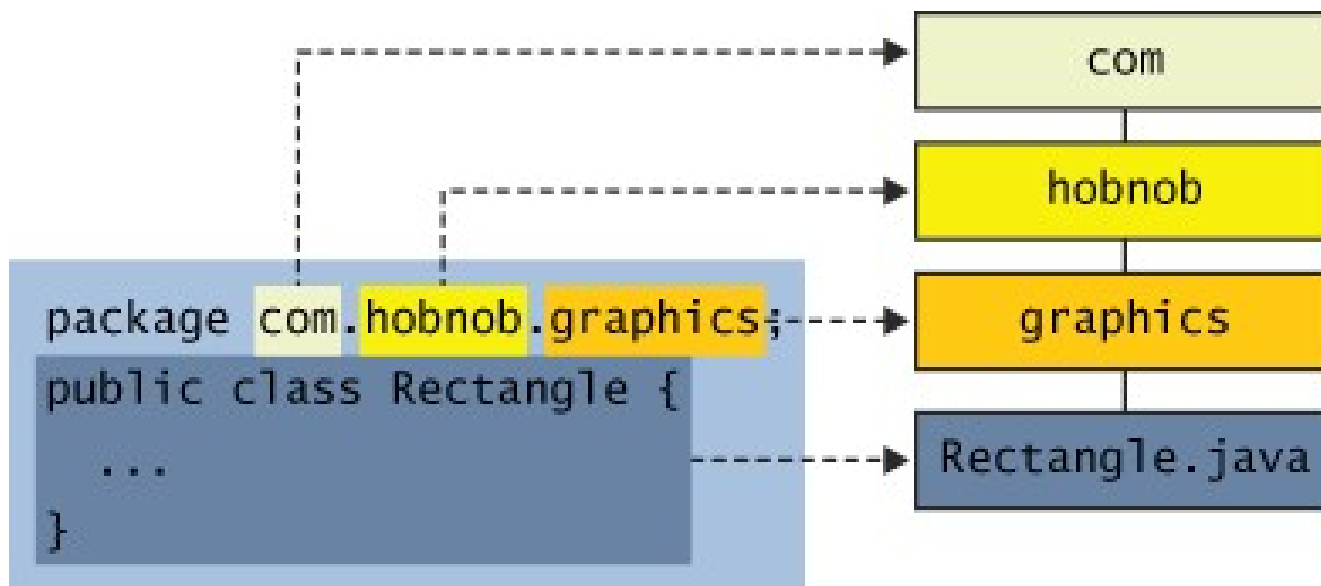
Gerenciamento de arquivos-fonte e de classes (cont.)

Companhias de desenvolvimento usam seus domínios Internet em reverso como nomes para seus pacotes;

Assim, uma companhia chamada hobnob.com definiria pacotes iniciados por `com.hobnob`.

Gerenciamento de arquivos-fonte e de classes (cont.)

- Se hobnob.com criasse o pacote `graphics` do exemplo, a estrutura de diretórios seria a seguinte:





Arquivos de saída (.class)

Ao ser compilado, um arquivo-fonte (.java) gera um arquivo de saída de mesmo nome para cada classe, interface etc, porém com a extensão .class.



Gerenciando `.java` e `.class`

Os arquivos `.class` também devem ficar em uma estrutura de diretórios refletindo seus nomes, mas não necessariamente junto dos arquivos-fonte.



Gerenciando `.java` e `.class`

Esta separação permite a disponibilização de programas aos usuários sem revelar o código-fonte:



O conceito de *class path*

Tanto o compilador quanto a máquina virtual Java (JVM) precisam saber exatamente onde estão as classes compiladas/invocadas pelo programa;

Por definição, o compilador e a JVM procuram por classes no diretório corrente e no arquivo JAR contendo os arquivos de classe da plataforma Java;

Outros diretórios podem ser especificados na variável de sistema `CLASSPATH`, que define uma lista ordenada de diretórios ou arquivos JAR onde classes devem ser procuradas.



Em resumo

Para criar um pacote, use a declaração `package` na primeira linha do código-fonte;
Ao usar uma classe de um pacote externo, há três opções:

- Referenciá-la pelo nome completo;
- Importá-la e usar o nome simples;
- Importar o pacote inteiro.

Pode ser necessário definir a variável `CLASSPATH` para que o ambiente Java encontre os arquivos criados.



Exercício

Suponha que você criou as classes abaixo no pacote default, e agora decidiu que elas devem ser organizadas em pacotes como segue:

Pacote	Classe
<code>mygame.server</code>	<code>Server</code>
<code>mygame.shared</code>	<code>Utilities</code>
<code>mygame.client</code>	<code>Client</code>

- (1) que linha de código adicionar a cada classe ?
- (2) quais diretórios criar e quais arquivos colocar em cada um ?
- (3) que outras alterações serão necessárias nestes arquivos ?



Respostas (1)

Em `Server.java`, adicionamos

```
package mygame.server;
```

Em `Utilities.java`, adicionamos

```
package mygame.shared;
```

Em `Client.java`, adicionamos

```
package mygame.client;
```



Respostas (2)

Dentro do diretório `mygame` criamos três subdiretórios: `server`, `shared` e `client`.

Em `mygame/server/` colocamos:
`Server.java`

Em `mygame/shared/` colocamos:
`Utilities.java`

Em `mygame/client/` colocamos:
`Client.java`



Respostas (3)

Para que cada classe (`Server`, `Utilities` e `Client`) possa “enxergar” uma a outra, é necessário que cada uma importe o que for necessário para a aplicação, ou faça referência aos nomes completos.



Pacotes no Eclipse

O ambiente Eclipse apresenta um conjunto de facilidades para gerenciamento de pacotes de forma automática;

Ao criar uma nova classe é possível especificar o pacote a qual ela pertence;

O Eclipse se encarrega da criação de diretórios etc.



Criando um novo projeto

Para criar um novo projeto:

Window Open Perspective Java

File New Project

Escolha “Java Project” da lista de opções;

Informe o nome do projeto e os demais dados.

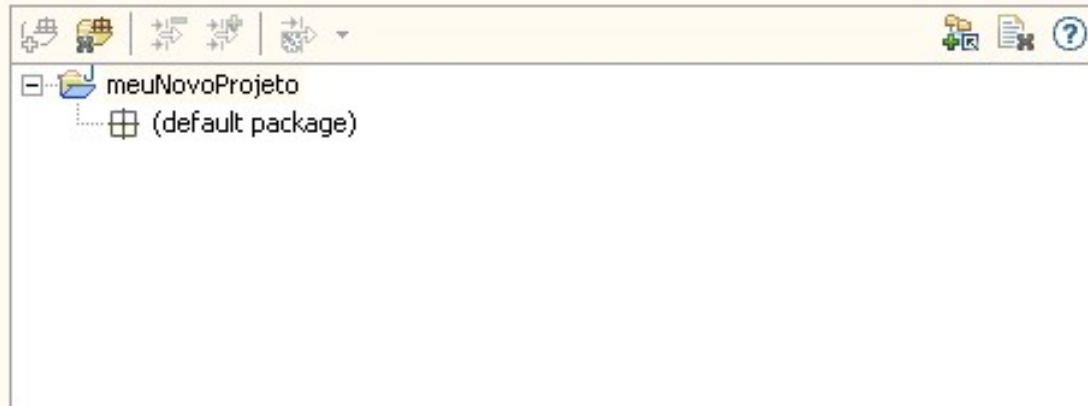


Java Settings


Define the Java build settings.



Source | Projects | Libraries | Order and Export



▼ Details

 Remove project 'meuNovoProjeto' from build path: Children of the project which are not source folders will not be seen by the compiler anymore and will not be included when building the project.

☐ Allow output folders for source folders

Default output folder:

meuNovoProjeto

Browse...

< Back

Next >

Finish

Cancel



Criando uma nova classe

File New Class

Certifique-se de especificar se deseja criar um método main;

Eclipse cria o “esqueleto” da classe conforme estipulado, já dentro de seu pacote (se for o caso).

New Java Class



Java Class

Create a new Java class.



Source folder:

meuNovoProjeto

Browse...

Package:

meuPacote

Browse...

☐ Enclosing type:

Browse...

Name:

MinhaClasse

Modifiers:

☒ public

☐ default

☐ private

☐ protected

☐ abstract

☐ final

☐ static

Superclass:

java.lang.Object

Browse...

Interfaces:

Add...

Remove

Which method stubs would you like to create?

☒ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments as configured in the [properties](#) of the current project?

☐ Generate comments

Finish

Cancel

```
package meuPacote;
```

```
public class MinhaClasse {
```

```
    /**
```

```
     * @param args
```

```
     */
```

```
    public static void main(String[] args) {
```

```
        // TODO Auto-generated method stub
```

```
    }
```

```
}
```